Mikroprozessorsystem K 1810 WM86

Hardware · Software · Applikation (Teil 1)

Prof. Dr. Bernd-Georg Münzer (wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann, Wolfgang Kabatzke, Frank Kamrad, Helfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock, Sektion Technische Elektronik, Wissenschaftsbereich Mikrorechentechnik/Schaltungstechnik

Das System K 1810 WM86 wird in dieser Kurs-Reihe in folgenden Komplexen umfassend vorgestellt:

1. Systemarchitektur
2. System-Schaltkreise
3. Interface-Schaltkreise
4. Interruptsystem
5. CPU-Assemblerbefehle
6. Assemblerprogrammierung mit SCP 1700
7. Coprozessoren
8. Programmentwicklung in C
9. Multitask-Verarbeitung
10. Systemüberblick

1. Systemarchitektur

Das 16-Bit-Mikroprozessorsystem K 1810 WM86 (8086) wird durch folgende Leistungsmerkmale gekennzeichnet:

- Datenbreite 16 Bit
- Speicheradreßraum 1 MByte
- E/A-Adreßraum 64 KByte
- CPU-Registerbreite 16 Bit
  - 4 Hauptregister
  - 2 Pointerregister
  - 2 Indexregister
  - 1 Flagregister
  - 4 Segmentregister
  - 1 Programmzähler
- CPU-Takt 5 MHz
- Coprozessorfähigkeit.

Es entspricht somit der mittleren Klasse seiner 16-Bit-Generation.
Das System 8086 hat international eine außerordentlich breite Anwendung gefunden, besonders auch durch den Einsatz in Personalcomputern führender Hersteller.
In den einzelnen Kapiteln dieses Kurses werden die Architektur, System- und Interface-Schaltkreise im Rahmen von applikativ erprobten Anwenderlösungen vorgestellt. Des weiteren folgen Beschreibungen zum Befehlssatz und zu Adressierungsmodi mit Beispielen zur Assemblerprogrammierung. Ein weiterer Abschnitt beschäftigt sich mit dem Multi- und Coprozessorverhalten. Danach werden die Möglichkeiten zur Programmentwicklung mit Hochsprachen einschließlich der Echtzeitprogrammentwicklung erläutert. Das abschließende Kapitel gibt einen Ausblick zu weiterentwickelten Systemen der oberen 16-Bit-Klasse.
Der Kurs hat sich zum Ziel gesetzt, in der gesamten Breite von Hardware und Software

Der Kurs hat sich zum Ziel gesetzt, in der gesamten Breite von Hardware und Software die Erfahrungen zum System 8086 zu vermitteln. Bezüglich von Detailbeschreibungen zu Parametern der Systemkomponenten, Vollständigkeit des Befehlssatzes und von Dienstprogrammen der Betriebssysteme wird auf die entsprechende Literatur der Entwickler hingewiesen.

1.1. Systemkomponenten

Eine 8086-Mikrorechner-Grundkonfiguration in der praxisrelevanten CPU-Betriebsart Maximum-Mode besteht aus folgenden Komponenten (Bild 1.1):

Systemschaltkreise

– 8284A Clock-Generator: Generierung des Systemtaktes CLK, READY- und RESET-Steuerung.
– 8086 CPU: Generierung des multiplexen Adreß-/Datenbus AD0 ... AD15, der höherwertigen Adressen A16 ... A19, der Statussignale $\overline{S0}$ ... $\overline{S2}$ und der Speicher-Bank-Steuerung Bus-High-Enable $\overline{BHE}$.
– 8288 Buscontroller: Generierung der Steuersignale für Speicher-($\overline{MRDC}$, $\overline{MWTC}$) und ($\overline{IORC}$, $\overline{IOWC}$), des Strobe-Signals ALE zur Adreßübernahme, der Freigabe- und Richtungssteuerung für die Datenbustreiber (DEN, DT/$\overline{R}$) und des Signales $\overline{INTA}$ zur Interruptbestätigung.

1.2. Busstruktur

Die 8086-CPU besitzt einen multiplexen Adreß-/Datenbus, der über Octal-Latches 8282 zum System-Adreßbus AB0 ... AB19 und über Datenbustreiber 8286 zum System-Datenbus DB ... DB15 geführt wird. Die Selektion von Daten und Adressen erfolgt mit Hilfe der System-Steuersignale

ALE address latch enable
DEN data enable
DT/R data transmit/receive.

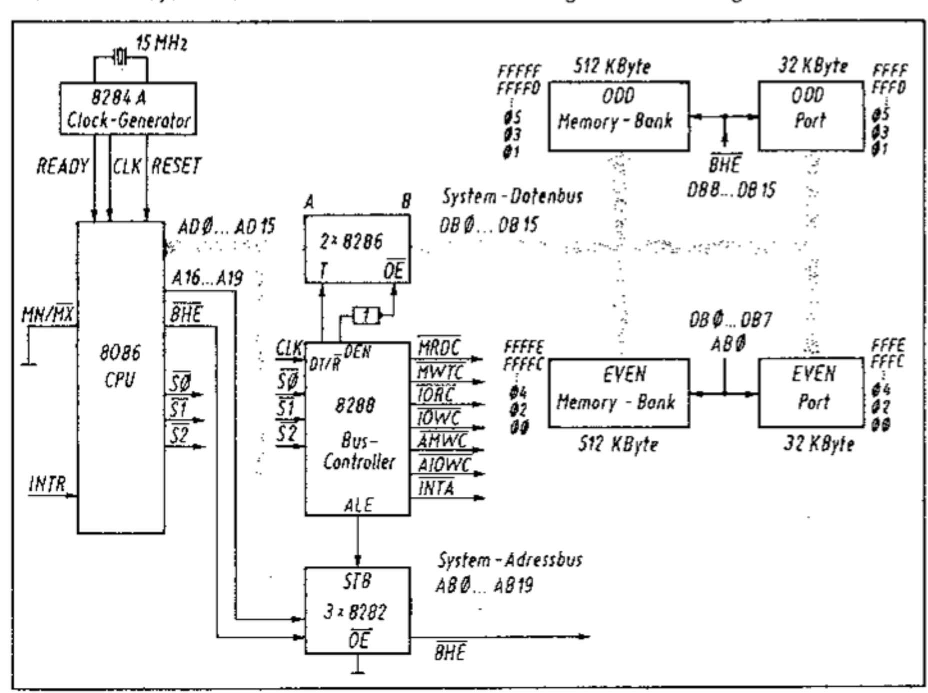
1.3. Speicher und E/A-Ports

Der Arbeitsspeicher ist byteorganisiert aufgebaut, das heißt, der gesamte Adreßraum von max. 1 MByte ist in 2 physisch getrennte 512-KByte-Banks (EVEN-, ODD-Bank) aufgeteilt. Die EVEN-Bank repräsentiert alle geraden Adressen (Chip-Select mit AB0 = 0), und die 8-Bit-Datenleitungen der Speicherbank sind zum niederwertigen Teil DB0 = DB7 des Systembusses geführt. Dementsprechend vereinigt die ODD-Bank alle ungeraden Adressen (Chip-Select mit $\overline{BHE}$ = 0), deren Datenleitungen mit DB8 ... DB15 verbunden sind. Die 8086-Ports mit dem maximalen Adreßraum von 64 KByte sind ebenfalls byteorganisiert und werden durch gerade/ungerade Adressen mittels AB0 und $\overline{BHE}$ selektiert (Bild 1.1). Die Ein-/Ausgabebefehle erlauben somit neben der Byte-Ein-/Ausgabe auch den Worttransfer über zwei 16-Bit-Ports benachbarter Adressen.

1.4. CPU Basis-Timing

Der 5-MHz-Systemtakt CLK, vom 8284A bereitgestellt, ist unsymmetrisch mit 1/3 High-Pegel und 2/3 Low-Pegel (Bild 1.2). Jeder CPU-Buszyklus besteht aus mindestens 4 Takten.
Vor dem Takt T1 jedes Buszyklus erfolgt die Bereitstellung der Statussignale S0, S1, S2 nach folgender Kodierung:

Bild 1.1. 8086-Systemarchitektur



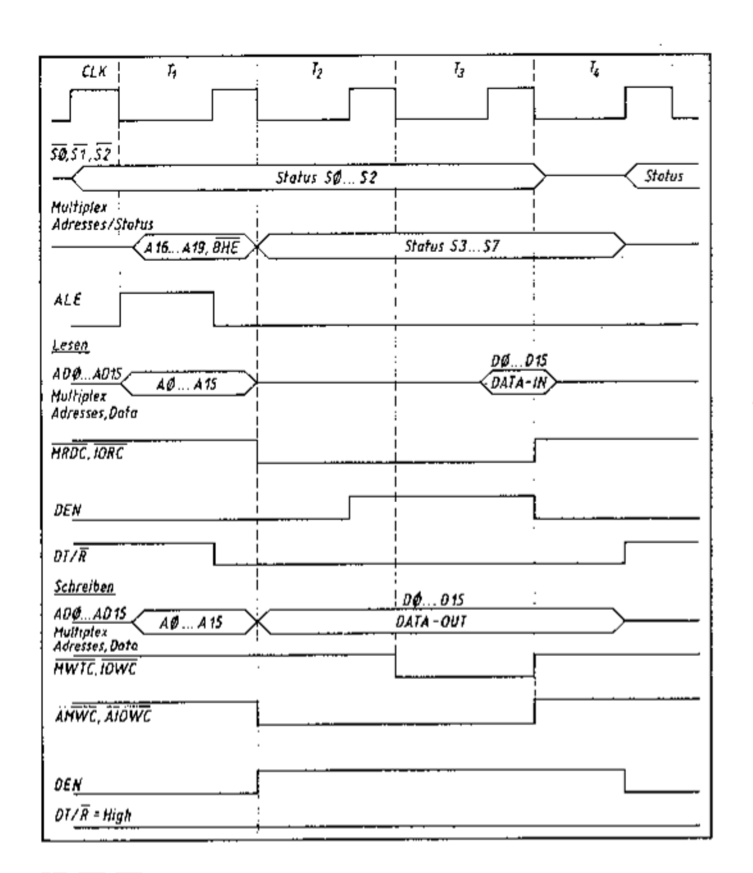


Bild 1.2 8086-Buszyklus

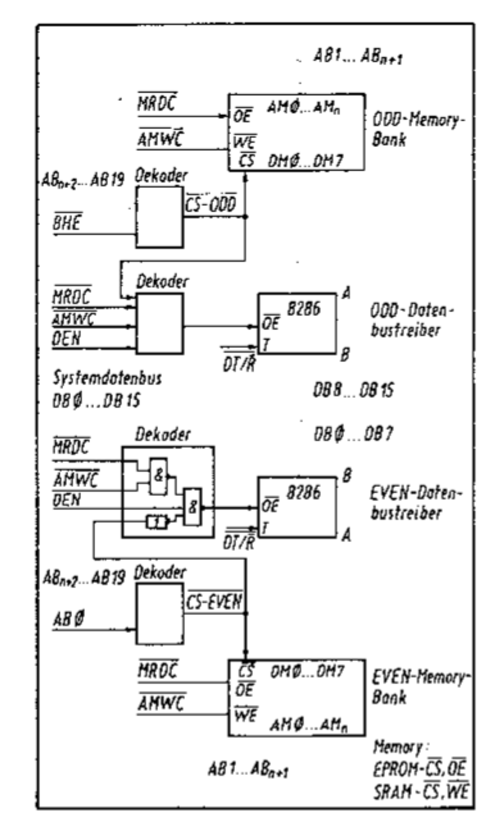


Bild 1.3 8086-Speicherarchitektur

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	Interrupt-Bestätigung
0	0	1	Porteingabe
0	1	0	Portausgabe
0	1	1	Halt
1	0	0	Befehlholen
1	0	1	Speicherlesen
1	1	0	Speicherschreiben
1	1	1	passiv, kein Buszyklus.

Aus diesen Statussignalen erzeugt der Buscontroller 8288 die Steuersignale

$\overline{MRDC}$ memory read control
$\overline{IORC}$ I/O read control
$\overline{MWTC}$ memory write control
$\overline{IOWC}$ I/O write control

und die zeitlich vorgezogenen Schreibsignale

$\overline{AMWC}$ advanced memory write control
$\overline{AIOWC}$ advanced I/O write control.

Zur Adreßübernahme in die Octal-Latches dient das Signal ALE zur Richtungssteuerung der Datenbustreiber DT/$\overline{R}$ und zur Freigabesteuerung der Datenbustreiber DEN.
Die Statussignale S3 ... S7 können durch eine extreme Logik gespeichert werden und enthalten Informationen über aktuelle Zugriffe zu Segmentregistern und zu den Zustandsbits.
Im Takt T1 werden von der CPU die Adressen A0 ... A15 (multiplex mit Daten), A16 ... A19, $\overline{BHE}$ (multiplex mit Status S3 ... S7) generiert und von den Octal-Latches mit der fallenden Flanke von ALE übernommen. Zu Beginn von T2 stehen damit
gültige Adressen und $\overline{BHE}$ bereit. Der Datentransfer wird durch Generierung der Steuersignale für die Speicher, für die E/A-Geräte und für die Datenbustreiber eingeleitet und mit den Takten T3/T4 abgeschlossen.

Speicherlese-/Befehlshole- und Portlesezyklen werden mit der Datenübernahme am Ende von T3 abgeschlossen. Bei Speicherschreib- und Portschreibzyklen werden unmittelbar nach T1 die Daten von der CPU bereitgestellt, um eine sichere Datenübernahme zu gewährleisten.

1.5. Speichertransfer

Der Datenaustausch zwischen CPU und Arbeitsspeicher erfolgt durch Aktivierung der Signale AB0, $\overline{BHE}$ byte- oder wortweise nach folgendem Modus:

$\overline{BHE}$	AB0	Transfer
1	0	Byte von gerader Adresse (EVEN-Bank) über DB0 ... DB7
0	1	Byte von ungerader Adresse (ODD-Bank) über DB ... DB15
0	0	Worttransfer gleichzeitig von EVEN- und ODD-Bank.

Befehlsholezyklen erfolgen grundsätzlich wortweise.
Die Adresse AB0 entspricht somit einem Freigabesignal für die EVEN-Bank (Bus-Low-Enable). Der 16-Bit-Transfer erfolgt mit AB0 = $\overline{BHE}$ = 0 parallel in einem Buszyklus, falls die niederwertige Adresse des Bytepaares geradzahlig ist. Wenn diese ungeradzahlig ist, wird der Worttransfer automatisch von der CPU in zwei Buszyklen zu je einem Byte zerlegt, zuerst mit AB0 = 1, $\overline{BHE}$ = 0 der ODD-Transfer und anschließend mit AB0 = 0, $\overline{BHE}$ = 1 der EVEN-Transfer.
Bild 1.3 zeigt einen Speichermodul mit EPROM und SRAM. Die Speicheradressen AM0 ... AMn werden mit AB1 ... ABn+1 des Systemadreßbus und die Datenleitungen DM0 ... DM7 jeweils mit dem niederwertigen Teil DB0 ... DB7 bzw. dem höherwertigen Teil DB8 ... DB15 des Systemdatenbus verbunden.

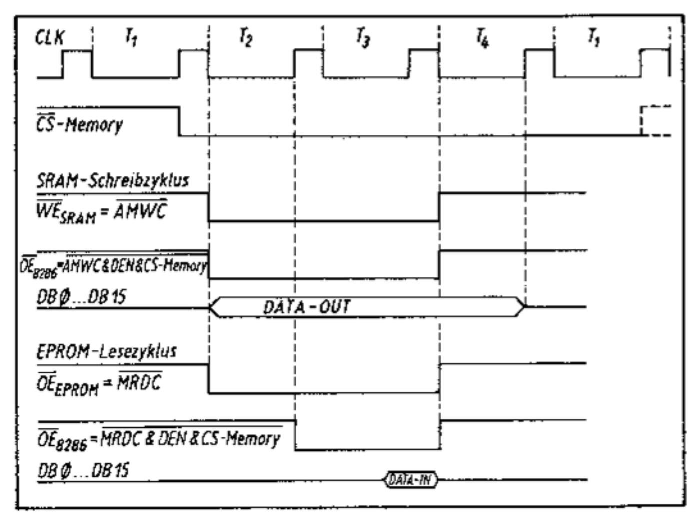


Bild 1.4 Speicherschreib-/-lesezyklus

Zur Sicherung der Datenzugriffszeiten wird die Speicher-CS-Steuerung nur aus den Adressen ABn + 2 . . . AB19 gebildet und für die EVEN/ODD-Bank mit AB0 bzw. $\overline{BHE}$ getort. Damit ist CS-Memory am Ende des Taktes T1 gültig (Bild 1.2). Im System 8086 dürfen nur Speicherschaltkreise mit pegelgesteuertem CS eingesetzt werden. Bei EPROMs wird der $\overline{OE}$-Eingang mit $\overline{MRDC}$ und bei SRAMs der $\overline{WE}$-Eingang mit $\overline{AMWC}$ verbunden.
Die Dekodierung für die OE-Freigabe-Steuerung der Datenbustreiber ist in Bild 1.3 im EVEN-Dekoder ausgeführt.
Die Richtungssteuerung T der Datenbustreiber erfolgt mit DT/$\overline{R}$.
Die Zeitbeziehungen beim Datentransfer am Beispiel eines SRAM-Schreibzyklus und eines EPROM-Lesezyklus sind in Bild 1.4 dargestellt.
Folgende Speicherschaltkreise können ohne WAIT-Zyklen bei einem Systemtakt von 5 MHz verwendet werden:

EPROM U2716 C 35
SRAM U2114 D 20.

Über die Anwendung dynamischer Speicher wird in einem späteren Abschnitt berichtet.
Eine praktikable Aufteilung des physischen 1-MByte-Speicherraumes in einem 8086-Rechner erfolgt nach folgendem Modus:

ROM

16 KByte	FC000H . . . FFFFFH	Monitor
	FFFF0H	Einsprung nach RESET

RAM

1/2 MByte	0400H . . . 7FFFFH	Daten, Stack und Anwenderbereich
1 KByte	0000H . . . 03FFH	Interrupt-Pointer-Tabelle

1.6. *Ein-/Ausgabetransfer*

Der Ein-/Ausgabetransfer über 8-Bit-/16-Bit-Ports wird äquivalent zum Speichertransfer nach Bild 1.1 und 1.3 mit folgenden systemeigenen Interface-Schaltkreisen realisiert:

8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
8255A Programmable Peripheral Interface (PPI)
8253 Programmable Interval Timer (PIT)
8259A Programmable Interrupt-Controller (PIC).

Die drei zuerst genannten Schaltkreise wurden aus dem 8-Bit-System 8080 übernommen. Diese Interface-Ports können keinen vektorisierten Interrupt auslösen, so daß ein im Interruptverhalten an die 8086-CPU angepaßter 8-Ebenen-Interrupt-Controller 8259A entwickelt wurde. Der PIC darf nur als EVEN-Port betrieben werden, da der Interruptvektor über den niederwertigen Systemdatenbus DB0 . . . DB7 von der CPU eingelesen wird.
Für diese Schaltkreise ist bei einem E/A-Zugriff jeweils ein Wait-Takt einzufügen.
Bei der Kopplung von U880-Interface-Schaltkreisen an das System 8086 ist eine Reihe von schaltungstechnischen Maßnahmen zur Kompatibilität des Steuerbus und des Interruptverhaltens erforderlich.

2. System-Schaltkreise

Die 8086-Systemschaltkreise sind:
–8086 – CPU
–8284A – Clockgenerator
–8288 – Buscontroller.

2.1. *CPU*

Die Standard-8086-CPU, mit 5 MHz Systemtakt und der Betriebsspannung + 5 V, wird in HMOS-Silicon-Gate-Technologie mit 29000 integrierten Transistoren in einem 40poligen DIL-Gehäuse hergestellt. (Bild 2.1).

2.1.1. CPU-Architektur

2.1.1.1. Funktionseinheiten
Die internen Funktionen des 8086-Mikroprozessors sind in zwei Einheiten aufgeteilt (Bild 2.2):
– Execution-Unit (EU)
– Bus-Interface-Unit (BIU).

Die EU enthält die grundsätzlichen Elemente einer CPU, wie Hauptregister, Arithmetic & Logic-Unit, Adreß- und Flagregister. Die EU führt die Dekodierung der von der BIU zwischengespeicherten Befehle durch. Nur die BIU besitzt Zugang zum externen 8086-Bus. Zur Erhöhung der Busbandbreite führt die BIU unabhängig von der EU ein vorausschauendes Holen von Befehlscodes und deren Zwischenspeichern in einem 6-Byte-FIFO-Instruction-Queue durch. Die Übergabe der Befehlscodes aus dem Instruction-Queue an die EU hängt von der Dekodierung und Ausführung des aktuellen Befehls ab, wobei die BIU dabei bemüht ist, den Instruction-Queue ständig gefüllt zu halten. Diese parallele und asynchrone Arbeit zwischen EU und BIU führt zu einer optimalen Busauslastung. Die bisher vom Anwender gewohnte sequentielle Folge von Maschinenzyklen auf einem Mikrorechnerbus in Korrespondenz mit den Maschinenzyklen der Befehle wird beim 8086-System verlassen (pipelining). Der aktuelle Transfer auf dem Systembus wird vom jeweiligen Zustand der EU (Befehlsdekodierung, -ausführung) und vom Instruction-Queue in der BIU (vorausschauendes Befehlsholen) bestimmt. Auf diese Tatsache muß der Anwender sich bei der Logikanalyse bzw. Busanzeige im READY/WAIT-Einzelschrittbetrieb einstellen.
Weiterhin erfolgt in der BIU die Generierung des 20-Bit-Adreßbus mit Hilfe der Segmentregister (vgl. Abschnitt 2.1.1.5.).

2.1.1.2. *Hauptregister*

Die CPU besitzt vier 16-Bit-Hauptregister, die vor allem bei arithmetisch/logischen und E/A-Operationen verwendet werden (Bild 2.3):
AX Accumulator
BX Base
CX Count
DX Data.
Diese Register sind auch jeweils als zwei 8-Bit-Register verwendbar und mit dem Index H bzw. L bezeichnet. Durch diese Aufteilung lassen sich auch Operationen mit 8-Bit-Operanden durchführen.

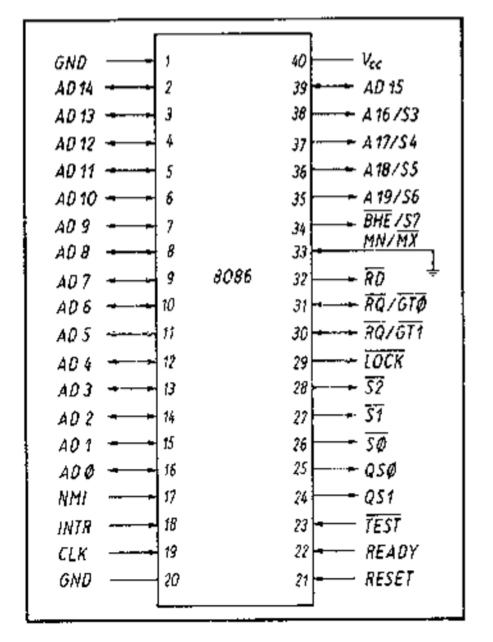


Bild 2.1 8086-CPU

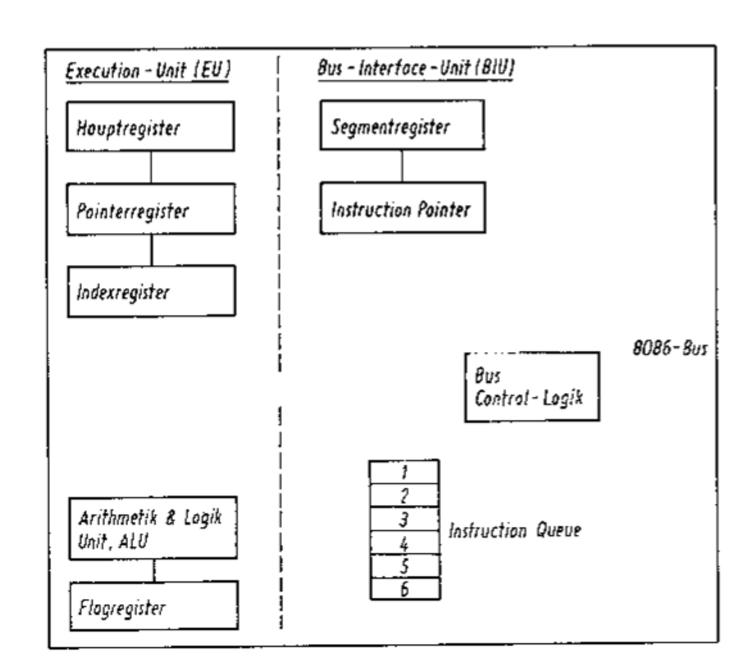


Bild 2.2 8086-CPU-Architektur

	15		Ø ein 16 Bit-Register
	7 High Ø	7 Low Ø	zwei 8 Bit-Register
AX	AH	AL	Accumulator
BX	BH	BL	Base
CX	CH	CL	Count
DX	DH	DL	Data

Bild 2.3 Hauptregister

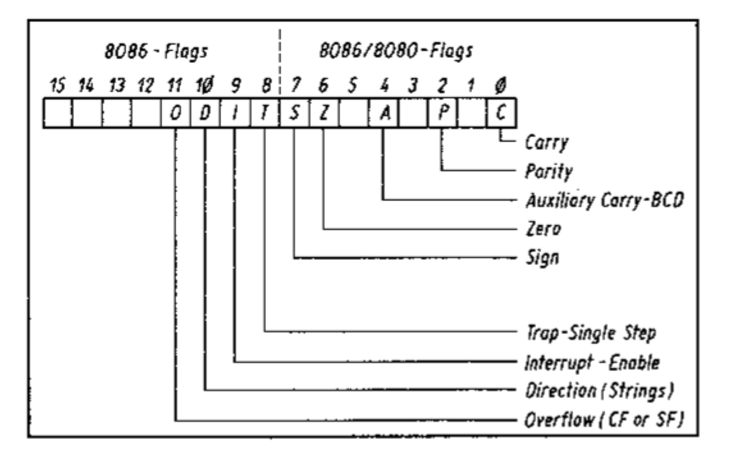


Bild 2.6 Flagregister

15 Ø	
SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index

Bild 2.4 Pointer- und Indexregister

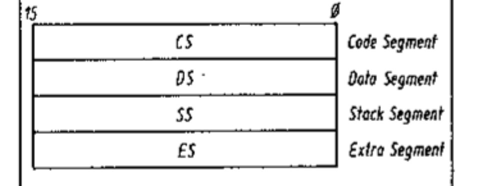


Bild 2.5 Segmentregister

Der Akkumulator AX wird vorzugsweise bei arithmetisch-logischen Operationen verwendet, und E/A-Befehle können nur über den Akkumulator ausgeführt werden. Das Base-Register BX dient zur Adressierung der Daten bei Speichertransfer-Operationen.
Das Count-Register CX wird bei LOOP- und Stringoperationen als Zählregister verwendet, und das Data-Register DX enthält bei einigen E/A-Operationen die 16-Bit-E/A-Adresse bzw. bei 16-Bit-Multiplikation/Division einen Teil des Ergebnisses.

2.1.1.3. Pointerregister

Über die 16-Bit-Pointerregister (Bild 2.4)

SP Stackpointer
BP Basepointer

werden Speicherplätze im Stacksegment adressiert. Der Stackpointer SP ermöglicht den Aufbau eines Stapels im Stacksegment. Mit dem Basepointer können zusätzliche Datentabellen im Stacksegment verwaltet werden.

2.1.1.4. Indexregister

Die 16-Bit-Indexregister (Bild 2.4)

SI Source-Index
SI Destination-Index

dienen zur Speicheradressierung bei Stringoperationen. Das SI-Register adressiert dabei den Quellbereich im Datensegment und das DI-Register den Zielbereich im Extrasegment.

Instruction-Pointer

Mit dem 16-Bit-Instruction-Pointer in der Bus-Interface-Unit erfolgt in Verbindung mit dem Code-Segment-Register die Adressierung von physischen Speicherplätzen bei Befehlsholezyklen.

2.1.1.5. Segment-Register

Die absolute Adressierung im 1-MByte-Speicherraum erfolgt mit Hilfe von 16-Bit-Segmentregistern (Bild 2.5):

CS Code-Segment
DS Data-Segment
SS Stack-Segment
ES Extra-Segment.

Jedes Segmentregister definiert einen 64-KByte-Bereich im Gesamtspeicherraum. Die Segmente müssen zu Beginn eines Programms geladen werden, und die Adreßbereiche können dabei auch überlappend festgelegt werden.
Das Code-Segment CS realisiert zusammen mit dem Instruction-Pointer die aktuelle Adresse des nächsten Befehls.
Das Stack-Segment SS realisiert z. B mit dem Stackpointer SP die aktuelle Adresse des Stapels bei PUSH, POP, CALL und RET-Operationen.
Mit dem Daten-Segment DS und z. B. dem BX-Register werden aktuelle Daten verwaltet.
Das Extra-Segment ES, als zusätzliches Datensegment, wird z. B. bei Stringoperationen eingesetzt.
Die Verwendung von Segmentregistern und Adreßregistern (IP, SP, BP, SI, DI, BX) erfolgt in definierten Zuordnungen, die im Abschnitt 2.1.3. erläutert werden.

2.1.1.6. Flagregister

Das 16-Bit-Flagregister (Bild 2.6) enthält im niederwertigen Teil 0 ... 7 die aus dem System 8080 übernommenen Zustandsbits, deren Bedeutung denen im U880-System entsprechen (außer P-Flag). Im höherwertigen Teil 8 ... 15 des Flagregisters sind folgende neue Zustandsbits aufgenommen.

T-Flag: Trap-Single Step

Dieses Zustandsbit wird beim Testen von Programmen eingesetzt und ermöglicht den softwaregesteuerten Einzelschrittbetrieb auch in ROM-Speicherbereichen. Wenn das T-Statusbit per Befehl auf 1 gesetzt wird, erfolgt am Ende des nächsten Befehls die Auslösung eines Software-Interrupts (vgl. Abschnitt Interrupt-Struktur).

I-Flag: Interrupt-Enable

Mit dem CPU-Befehl Interrupt-Enable wird das I-Flag = 1 gesetzt und damit der maskierbare Interrupt zugelassen. Nach einer Interruptannahme oder nach Ausführung des CPU-Befehls Interrupt-Disable erfolgt ein Rücksetzen des I-Flag und damit Interruptsperre.

D-Flag: Direction

Das D-Flag beeinflußt die Richtung des Datentransfers bei Stringoperationen. Bei Setzen des D-Flag = 1 mit dem CPU-Befehl Set-Direction erfolgt bei Stringoperationen automatisch ein Adressendekrement der Indexregister SI, DI. Mit dem CPU-Befehl Clear-Direction (D = 0) wird ein Auto-Inkrement von SI und DI realisiert.

O-Flag: Overflow

Bei arithmetisch-logischen Operationen wird das Carry-Bit in einer Exclusive-Oder-Verknüpfung mit dem Übertrag von Bit 6 nach Bit 7 zum Overflow-Flag verknüpft. Wenn das Overflow-Flag im Ergebnis einer Operation gesetzt und anschließend der Software-Interrupt-Befehl INTO ausgeführt wurde, erfolgt die Durchführung einer Interrupt-Service-Routine (vgl. Abschnitt Interrupt-Struktur).

2.1.2. Elektrische Signale und Anschlüsse

Die Anschlüsse der CPU (Bild 2.1) sollen nur in der Betriebsart Maximum-Mode (Anschluß MN/$\overline{MX}$ = 0) erläutert werden; die Betriebsart Minimum-Mode hat sich in der applikativen Praxis nicht durchgesetzt.
Die CPU-Signale können in die Kategorien
– Adreßsignale
– Datensignale
– Statussignale
eingeteilt werden. Eine Reihe der Anschlüsse ist zeitmultiplex ausgelegt.

Adreß- und Datensignale/Statussignale AD0 ... AD15 (input, output, tristate)

Im Takt T1 (vgl. Bild 1.2) werden bei Speicher- und E/A-Operationen die Adressen A0 ... A15 ausgegeben, in den nachfolgenden Takten T2 und T3 erfolgt der Datentransfer D0 ... D15.

A16/S3; A17/S4; A18/S5; A19/S6 (output, tristate)

An diesen 4 Anschlüssen werden bei Speicheroperationen im Takt T1 die höherwertigen Adressen A16 ... A19 gültig, bei E/A-Operationen sind diese Ausgänge low. Bei Speicher und E/A-Operationen sind während der Takte T2, T3, T4 die Statusinformationen S3, S4, S5 und S6 aktiv.
In S3 und S4 ist der Zugriff auf die Segmentregister im aktuellen Buszyklus nach folgender Vorschrift kodiert:

S4	S3	Bedeutung
0	0	Extra-Segment
0	1	Stack-Segment
1	0	Code-Segment oder kein Segment
1	1	Datensegment.

Die Statussignale enthalten folgende Angaben:
S5 Wert des Interrupt-Enable-Flag
S6 = 0; CPU ist aktueller Bus-Master
= tristate; CPU hat Buskontrolle abgegeben

$\overline{BHE}$/S7 (output, tristate)

Im Takt T1 erfolgt die Ausgabe des Speicherbank-Signales $\overline{BHE}$, der Status S7 in den Takten T2 ... T4 ist nicht näher definiert. Die Statussignale S3 ... S4 werden von den systemeigenen Schaltkreisen nicht ausgewertet.

$\overline{S0}$, $\overline{S1}$, $\overline{S2}$ (output, tristate)

Die Statussignale $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ geben Informationen für den Buscontroller entsprechend der Kodierung von Abschnitt 1.

QS0, QS1 (output)

Die Statusbits QS0, QS1 werden von den Coprozessoren ausgewertet und geben Auskunft über den aktuellen Zustand im Zwischenspeicher des CPU-Instruction-Queue.

QS1	QS0	
0	0	keine Operation
0	1	erstes Byte eines Befehls wurde dem Instruction-Queue entnommen
1	0	Instruction-Queue ist leer
1	1	ein nachfolgendes Byte wurde dem Instruction-Queue entnommen

Diese Statusbits gelten während des Taktes nach einer Operation im Instruction-Queue.

Steuersignale
RESET (input, output, high-aktiv)

Das RESET-Signal, vom Clockgenerator 8284A einsynchronisiert, führt ein Rücksetzen der CPU durch. Danach stellt sich in der CPU folgender Zustand ein:
– Inhalt des Code-Segments = FFFFH
– Inhalt des Instruction-Pointers = 0000
damit lautet die physische Adresse des ersten Befehls FFFF0H
– Inhalt von Daten-, Stack- und Extra-Segment = 0000
– Rücksetzen aller Bits des Statusregisters (Interruptsperre, kein single-step).

INTR (input, high-aktiv)

Die pegelaktive Interruptanforderung wird im letzten Takt jedes Befehls abgefragt. Falls Interruptfreigabe gesetzt wurde und an INTR High-Pegel anliegt, erfolgt ein Interrupt-Annahmezyklus mit Generierung des Interrupt-Bestätigungssignals INTA durch den Buscontroller 8288 (vgl. Abschnitt Interrupt-Struktur).

NMI (input, high-aktiv)

Der flankengetriggerte NMI für den nichtmaskierbaren Interrupt wird im letzten Takt eines jeden Befehls abgefragt. Die Startadresse für die Interrupt-Service-Routine liest die CPU aus den Speicherplätzen 0008H ... 000BH.

READY (input, high-aktiv)

In Verbindung mit dem Clockgenerator 8284A erfolgt die WAIT-Steuerung der CPU (vgl. Abschnitt 2.2).

$\overline{TEST}$ (input, low-aktiv)

Nach Dekodierung eines WAIT-Befehls fragt die CPU den $\overline{TEST}$-Eingang ab. Es werden so lange keine weiteren Befehle eingelesen, bis $\overline{TEST}$ = low wird (vgl. Abschnitt Coprozessoren).

$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ (input, output)

Die $\overline{RQ}/\overline{GT}$-Anschlüsse (0 = höhere, 1 = niedere Priorität) werden bei Anwesenheit weiterer Master im Local-Bus benutzt (DMA-Betrieb, Coprozessoren).
Die Anschlüsse sind bidirektional ausgeführt, und durch eine zeitlich genau definierte Folge von 3 Impulsen wird der Bus-Annahme- und Bus-Rückgabezyklus durchgeführt (vgl. Abschnitt Coprozessoren).

$\overline{LOCK}$ (output, tristate)

Ein Präfix LOCK vor einem Befehl bewirkt, daß während des nächsten Befehls der $\overline{LOCK}$-Ausgang aktiv = low wird. Damit wird anderen Coprozessoren mitgeteilt, daß während dieses geschützten Befehls keine Busübergabe stattfinden darf (vgl. Abschnitt Coprozessoren).

CLK (input)

Der CLK-Eingang der CPU ist mit dem entsprechenden Ausgang des Clockgenerators 8284A zu verbinden und realisiert den Systemtakt.

$\overline{RD}$ (output, tristate)

Ein Low-Pegel an diesem Ausgang signalisiert einen Speicherlese- oder Eingabezyklus. In Maximum-Mode wird dieses Signal kaum verwendet.

2.1.3. Speicheradressierung

Die Generierung der 20-Bit-physischen Speicheradresse im System 8086 erfolgt durch Addition von 2 Bestandteilen:
– 16-Bit-Basisadresse (Inhalt eines der Segmentregister CS, DS, SS, ES)
plus
– 16-Bit-Effektive-Adresse (Offset, Inhalt eines der Adreßregister IP, SP, BP, SI, DI, BX).

Die Bus-Interface-Unit BIU führt automatisch die Addition von Basis- und Offsetadresse nach folgendem Modus durch (Bild 2.7).

– Verschiebung des Inhaltes des Segmentregisters um 4 Bit-Positionen nach links, Auffüllung der Tetrade mit 0000
– Addition mit der Effektiven Adresse.

Ein Beispiel zur Realisierung einer konkreten physischen Adresse ist in Bild 2.8 dargestellt. Durch diese Aufteilung von Basis- und Offsetadresse wird der 1-MByte-Speicherraum in vier frei wählbare Segmente zu je 64 KByte

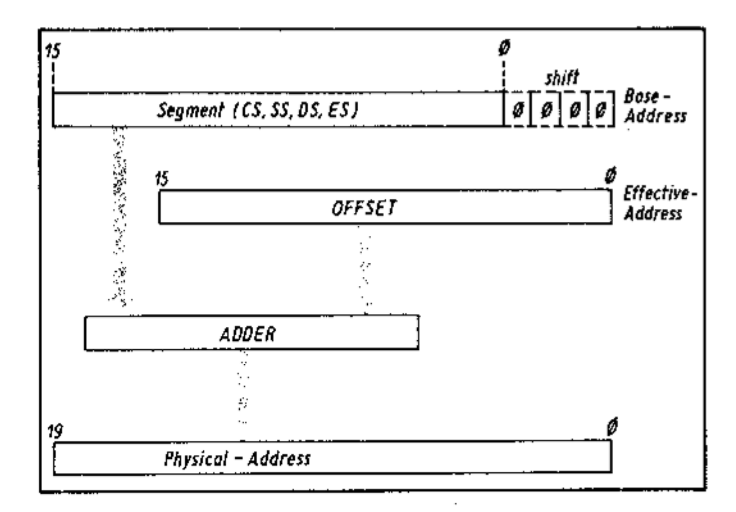


Bild 2.7 Adreßbildung mit Segment-Registern

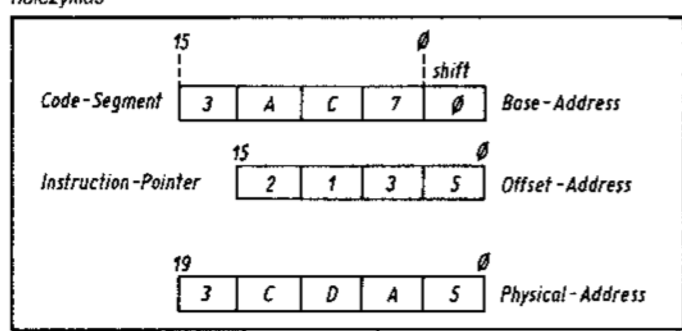


Bild 2.8 Beispiel zur Bildung der physikalischen Adresse bei einem Befehlsholezyklus

aufgeteilt. Die Segmentregister können eine beliebige Zahl enthalten und zeigen auf den Nullpunkt eines 64-KByte-Segment-Speicherraumes. Eine absolute Adresse ist nun innerhalb dieses Segment-Speicherraumes festgelegt. Der Abstand der Segment-Nullpunkte zueinander ist n × 16 Byte (n = 0,1 . . . 64K). Durch Einführung der Segmentierung wird auch eine klare logisch-physische Trennung von Programmen (CS im ROM), Daten (DS und ES im RAM) und Stack (SS im RAM) erreicht. Im 8086-Befehlssatz mit seinen Adressierungsmodi ist eine Standard-Zuordnung der Register für die Bildung der Effektiven Adresse (Operanden-Register) mit den Segmentregistern festgelegt (Bild 2.9). Eine unveränderliche Zuordnung haben folgende Operandenregister:

- Instruction-Pointer, IP
- Stack-Pointer, SP
- Destination-Index, DI.

Damit sind folgende konstante Beziehungen realisiert:

1. Befehlsholezyklen werden grundsätzlich im Code-Segment zusammen mit dem Instruction-Pointer durchgeführt.
2. Alle Befehle, die den Stack-Pointer benutzen (PUSH, CALL usw.), werden grundsätzlich im Stacksegment wirksam.
3. Stringbefehle, die bei der Adreßbildung den Inhalt des Destination-Index-Registers verwenden, entnehmen die Basisadresse immer dem Extrasegmentregister.

Bei der Adressierung mit Hilfe der Operanden-Register

- Base-Pointer, BP
- Base-Register, BX
- Source-Index, SI

kann durch ein Override-Präfix (1 Byte) vor dem Befehl die feste Zuordnung aufgehoben und ein beliebiges anderes Segmentregister zur Bildung der Basisadresse herangezogen werden (z. B. JMP CS: BP, Sprung auf eine Adresse im Codesegment, deren Offset durch den Inhalt von BP festgelegt ist).
Die Anwendung der Operandenregister zur Adressierung wird im Abschnitt Adressierungsmodi genauer erläutert.

2.2. *Clockgenerator 8284A*

Der Clockgenerator 8284A in der 8086-Systemarchitektur nach Bild 1.1 besitzt mit den Anschlüssen nach Bild 2.10 die Funktionseinheiten (Bild 2.11)

- Takterzeugung
- READY/WAIT-Steuerung
- RESET-Steuerung.

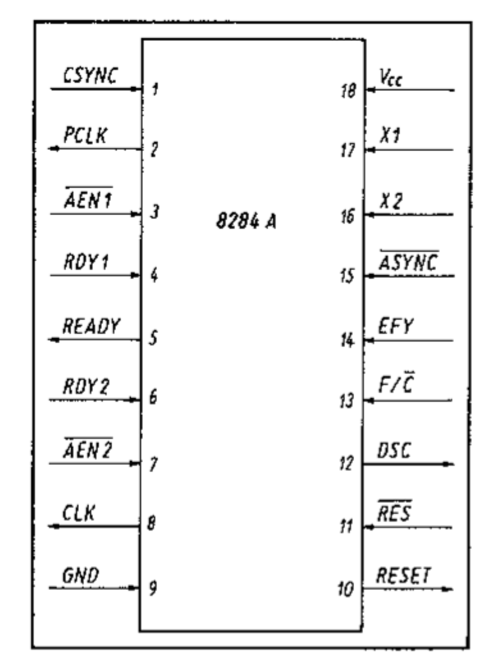


Bild 2.10 8284A-Clockgenerator

2.2.1. Takterzeugung

Die Funktion der Takterzeugung ist in Bild 2.11 dargestellt. Die Frequenz des Quarz-Oszillators wird zur Generierung von CLK einem 1:3-Teiler zugeführt. PCLK entsteht durch 1:2-Teilung aus CLK.
Zur Takterzeugung im System 8086 (Bild 2.10) werden folgende Anschlüsse des 8284A benötigt:

X1, X2 Chrystal-in (input)

An diese Pins wird der externe Quarz zur Erzeugung des Systemtaktes CLK angeschlossen. Die Quarz-Frequenz ist 3mal größer als die erforderliche Systemtakt-Frequenz.

F/$\overline{C}$ Frequency/Chrystal Select (input)

Wenn dieser Eingang statisch gleich low gelegt wird, erfolgt die Realisierung des Systemtaktes durch den an X1, X2 angeschlossenen Quarz. Wenn F/$\overline{C}$ gleich high ist, wird CLK durch eine an das pin EFI anzulegende externe Taktversorgung generiert.

EFI External Frequency (input)

Die externe Frequenz muß 3mal größer sein als die erforderliche Frequenz des Systemtaktes.

CSYNC Clock Synchronization (input, high-aktiv)

Der CSYNC-Eingang wird bei externer Taktversorgung mit EFI-pin benötigt und erlaubt, mehrere 8284A-Clockgeneratoren miteinander zu synchronisieren. Wenn nur ein Clockgenerator mit Quarz verwendet wird, ist CSYNC mit Masse zu verbinden.

CLK Prozessor Clock (output)

Der unsymmetrische Systemtakt CLK mit der Frequenz gleich 1/3 Quarzfrequenz dient zur Taktversorgung für CPU und Buscontroller. Der H-Pegel (4,5 V) beträgt 1/3 und der L-Pegel 2/3 der Zyklusperiode (Bild 1.2).

PCLK Peripheral Clock (output)

Der Peripherietakt PCLK ist ein symmetrisches Signal mit der Frequenz gleich 1/2 CLK

Bild 2.11 8284A-Funktionsbild

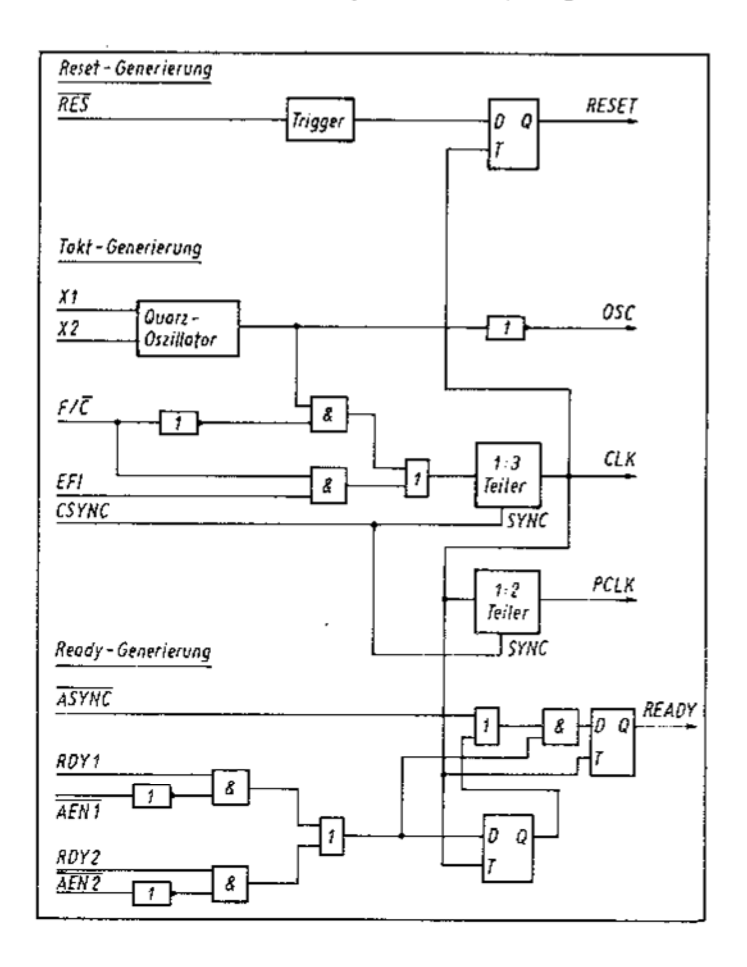


Bild 2.9 Zuordnung der Operanden-Register zu den Segmenten

Effektive Adresse mit Operanden-Register	Standard-Zuordnung Segment-Register	Alternative Segmente mit Override-Prefix
Instruction-Pointer, IP	Code-Segment, CS	nein
Stack-Pointer, SP	Stack-Segment, SS	nein
Base-Pointer, BP	Stack-Segment, SS	ja
Base-Register, BX	Data-Segment, DS	ja
Source-Index, SI	Data-Segment, DS	ja
Destination-Index, DI	Extra-Segment, ES	nein

und TTL-Pegel. PCLK wird für die Taktversorgung einiger Interface-Schaltkreise verwendet (z. B. 8251A-USART).

OSC Oscillator (output)

Der Takt OSC besitzt die Quarz-Frequenz mit TTL-Pegel und dient zum Treiben weiterer Clockgeneratoren an deren Eingang EFI. Eine applikative Lösung zur Realisierung des praxisrelevanten Systemtaktes 4,9152 MHz und damit der doppelten K-1520-Taktfrequenz ist in Bild 2.12 dargestellt. Die Serienwiderstände 510 Ohm an den Eingängen X1, X2 sind zur Stabilitätssicherung zwingend erforderlich.

2.2.2. READY-Synchronisation

Das System 8086 ist so ausgelegt, daß bei jedem Datentransfer eine Bestätigung von den Speichern bzw. E/A-Einheiten an die CPU erfolgt. Die Synchronisation der peripheren Bestätigung mit dem CPU-Buszyklus erfolgt im Clockgenerator 8284A durch folgende Anschlüsse (vgl. Bild 2.10, Bild 2.11):

RDY1, RDY2 Bus Ready-Transfer complete (input, high-aktiv)

Die Realisierung der Busbestätigung in Multi-Master-Systemen erfolgt an gleichwertigen Bestätigungseingängen RDY1, RDY2. Ein aktiver High-Pegel an RDY signalisiert von den peripheren Einheiten, daß die Daten empfangen bzw. gesendet wurden und der RUN-Betrieb der CPU erfolgen kann.

$\overline{AEN1}$, $\overline{AEN2}$ Access-Enable (input, low-aktiv)

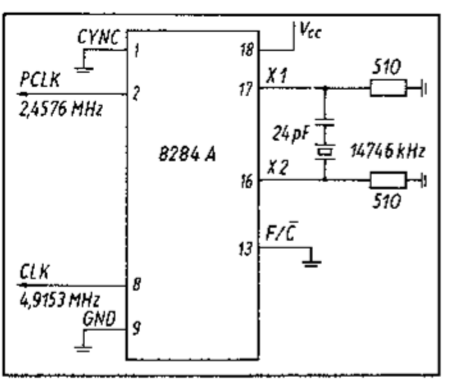
Jeder der Bus-Ready-Eingänge RDY1, RDY2 besitzt ein Torsignal $\overline{AEN1}$, $\overline{AEN2}$. Entsprechend Bild 2.11 erfolgt eine UND-Verknüpfung von RDY1&AEN1 bzw. RDY2&AEN2, die Ausgänge der UND-Gatter sind über eine ODER-Verknüpfung und Synchronisation mit CLK zum Ausgang READY geführt.

READY (output, high-aktiv)

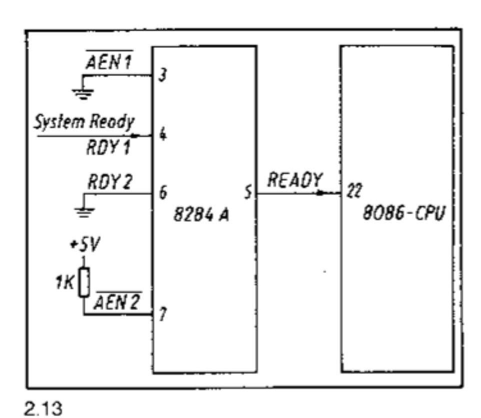
Der READY-Ausgang des Clockgenerators 8284A wird mit dem entsprechenden Eingang der 8086-CPU verbunden. Ein High-Pegel an READY-OUT-8284A gibt die synchronisierte Information für den RUN-Betrieb an die CPU weiter. READY gleich Low versetzt die CPU in einen WAIT-Zustand. Die Beschaltung des Clockgenerators 8284A zur Realisierung des RUN-Modus (System-Ready gleich High) und des WAIT-Modus (System-Ready gleich Low) über RDY1, $\overline{AEN1}$ ist in Bild 2.13 dargestellt (logische Verknüpfung von Bild 2.11 beachten!).

Wirkungsabläufe
RUN-Modus

Zur Realisierung des CPU-RUN-Modus ohne Warteschritte benötigt die CPU (vgl. Bild 2.14) ein high-aktives READY-IN-Signal mit der Set-up-Zeit von 118 ns vor der steigenden Flanke im Takt T 3. Die Übernahme der Bestätigungssignale RDY, $\overline{AEN}$ in den Clockgenerator 8284A erfolgt mit CLK am Ende von T 2. Zur internen Synchronisation müssen RDY, $\overline{AEN}$ mit den angegebenen Set-up-Zeiten aktiviert werden, um einen sicheren RUN-Betrieb zu gewährleisten. Für



2.12

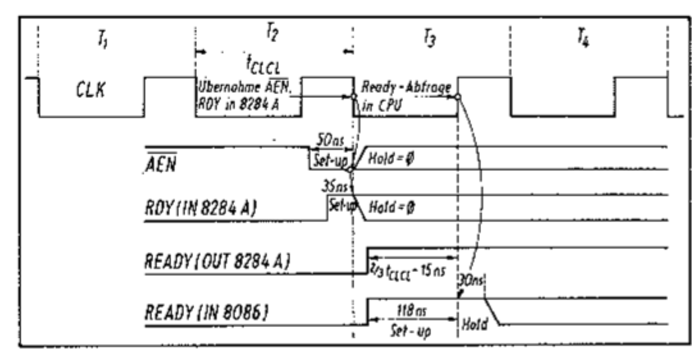


2.13

Bild 2.12 Realisierung eines 5 MHz-Systemtaktes

Bild 2.13 System-Ready mit RDY1, $\overline{AEN1}$

Bild 2.14 Zeitverläufe im CPU-RUN-Modus



2.14

einen permanenten RUN-Modus ist in Bild 2.13 RDY1 konstant gleich high zu legen.

WAIT-Modus

Die pins RDY, $\overline{AEN}$ können auch zur Realisierung eines WAIT-Modus verwendet werden, was am Beispiel der Einfügung eines WAIT-Schrittes in jeden Buszyklus erläutert werden soll (Bild 2.15).
Das Eingangssignal RDY (mit $\overline{AEN}$ gleich low) wird mit der Set-up-Zeit 35 ns vor dem Ende von T 2 inaktiv gleich low gesetzt (WAIT). Diese Information wird mit der steigenden Flanke von T 3 durch die CPU an READY abgefragt und nach T 3 ein WAIT-Zyklus Tw eingefügt. Wenn RDY vor dem Ende T 3 wieder aktiv gleich high gesetzt wird, dann folgt nach dem WAIt-Schritt Tw der Takt T 4, und der Buszyklus wird beendet.
In der applikativen Praxis ist der Generierung des RDY-Signals besondere Aufmerksamkeit zu widmen. Infolge von Verzögerungszeiten der verwendeten zusätzlichen Logikschaltkreise müssen erhebliche Toleranzgrenzen für die angegebenen Zeitbeziehungen eingehalten werden. Zur Erzeugung eines oder mehrerer WAIT-Schritte muß daher RDY bereits im Takt T 1 gleich low sein, also z. B. aus den Systemsignalen ALE oder $\overline{S0}$, $\overline{S1}$, $\overline{S2}$; AB0 ... AB19 abgeleitet werden (vgl. Bild 2.15). Eine Verknüpfung von RDY mit den Steuersignalen $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{AIOWC}$, $\overline{INTA}$ des Buscontrollers 8288 ist zur WAIT-Auslösung nicht möglich.

2.2.3. RESET-Steuerung

Die Synchronisation des RESET-Timings erfolgt über den $\overline{RES}$-Eingang des 8284A zum RESET-Eingang der 8086-CPU (Bild 2.16). Die CPU benötigt ein mindestens 4 Takte langes high-aktives Eingangssignal RESET. Der $\overline{RES}$-Eingang im 8284A wird über einen Schmitt-Trigger und Synchronisation mit CLK zum RESET-OUT-8284A geführt, der von der CPU abgetastet wird. Während des RESET-Timings sind die Signale des 8086-Bussystems inaktiv bzw. tristate. Die Statussignale $\overline{S0}$, $\overline{S1}$, $\overline{S2}$, die zuerst passiv-, dann tristate-Verhalten aufweisen, sind über interne pull-up-Widerstände im Buscontroller 8288 geführt, so daß entsprechend Bild 1.1 das gesamte 8086-System während des RESET-Timings einen inaktiven Zustand einnimmt. Eine applikative Lösung ohne power-on-RESET zeigt Bild 2.17.

2.3. Bus Controller 8288

Der bipolare Bus Controller 8288 (Bild 2.18) im 20-pin-DIL-Gehäuse generiert aus den CPU-Steuersignalen $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ den Steuerbus des 8086-Systems (vgl. Abschnitt 1.4. und Bild 1.1).
Für die Anwendung in System-Bus-Mode in der Konfiguration von Bild 1.1 sind folgende Anschlüsse von Bedeutung:
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$ Status (input)
Diese Eingangsleitungen werden mit den entsprechenden Signalen der 8086-CPU verbunden. Die Kodierung des Status ist in Abschnitt 1.1. dargestellt.

CLK Clock (input)
Systemtakt vom 8284A-Clockgenerator

IOB Input/Output Bus Mode (input)
$\overline{AEN}$ Address Enable (input)
In System-Bus-Mode nach Bild 1.1 sind die Eingänge IOB, $\overline{AEN}$ auf Masse zu legen.

CEN Command Enable (input)
Wenn der CEN-Eingang auf Low gelegt wird, dann nehmen die 8288-Command-Outputs den tristate-Zustand ein (DMA-Betrieb). Im Single-CPU-Modus ist der CEN-Eingang auf High-Potential zu legen.

Bild 2.15 Zeitverläufe im CPU-WAIT-Modus mit einem WAIT-Zyklus

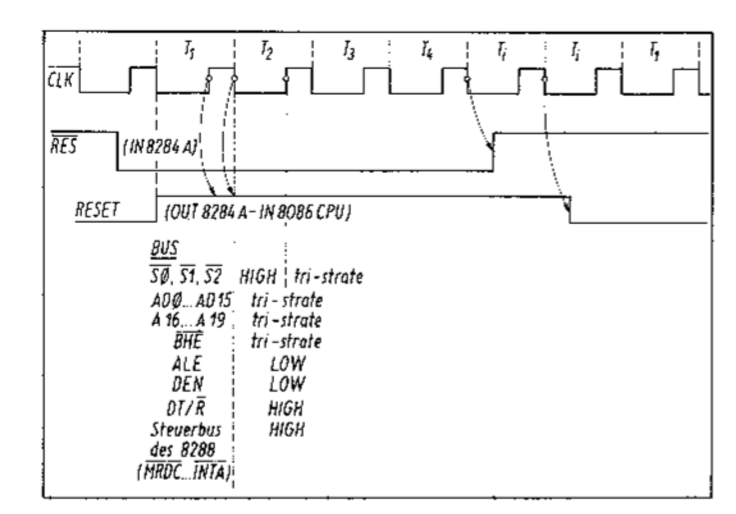


Bild 2.16 RESET-Timing

Die Command Outputs (Iol = 32 mA)
- $\overline{MRDC}$;
- $\overline{MWTC}$;
- $\overline{IORC}$;
- $\overline{IOWC}$;
- $\overline{AMWC}$;
- $\overline{AIOWC}$;
- $\overline{INTA}$

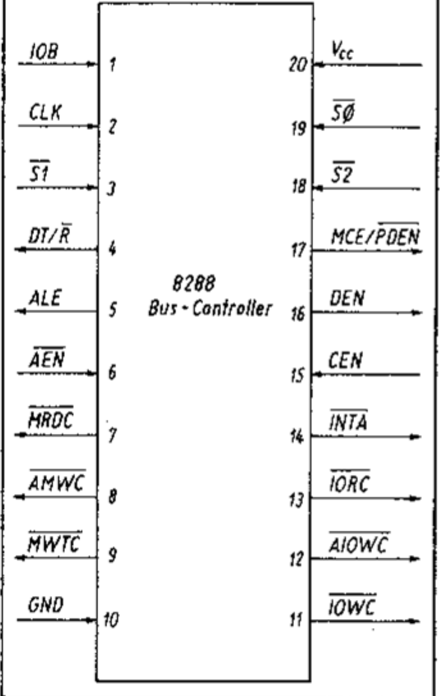
werden zum Steuerbus des Systems 8086 geführt.

Die Control Outputs (Iol = 16 mA)
- ALE;
- DEN;
- DT/$\overline{R}$

stellen die Steuersignale für die Address-Latches und die Bustreiber dar. Der zeitliche Verlauf von Command- und Control-Outputs ist in Bild 1.2 dargestellt. Die Beschaltung des Bus Controllers 8288 für eine Systemarchitektur nach Bild 1.1 ist in Bild 2.19 vorgestellt.

Literatur

/1/ MCS-86 User's Manual, Intel-Corp.
/2/ R. Rector, G. Alexy: Das 8086/8088-Buch, te-wi Verlag, 1982



2.18

Termine

Fachtagung Neue Erkenntnisse Mikroelektronik/Mikrorechentechnik

WER? Bezirksverband Halle der KDT

WANN? 3. und 4. Mai 1988

WO? Wittenberg, Kreiskulturhaus „Maxim Gorki"

WAS?
- 16-Bit-Prozessortechnik
- Bauelemente für Mikrorechner und periphere Baugruppen
- Stromversorgung von mikroelektronischen Baugruppen und Geräten
- Softwaretechnologie

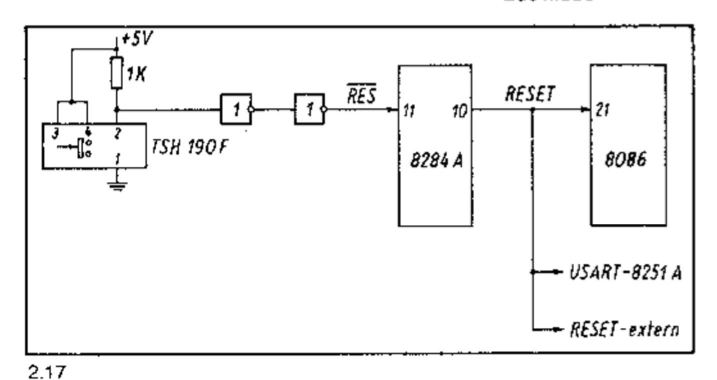
WIE? Anfragen an Kammer der Technik, Bezirksverband Halle, Geschwister-Scholl-Str. 39, Halle, 4030

Große

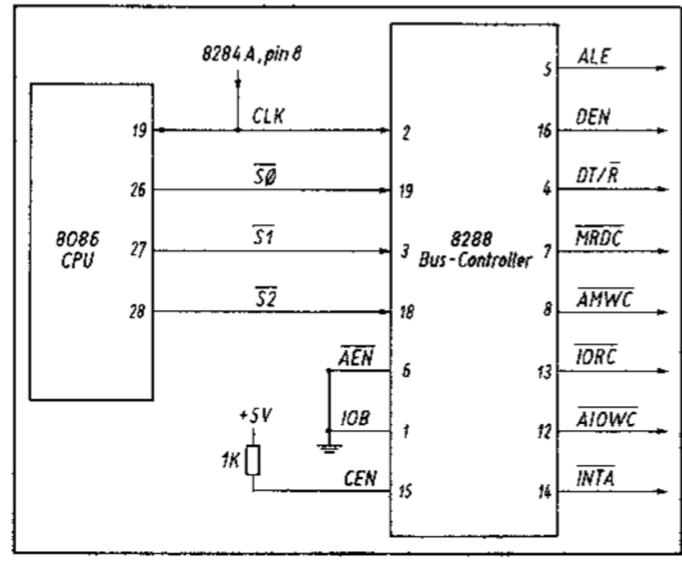
Bild 2.17 RESET-Generierung

Bild 2.18 Bus Controller 8288

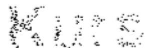
Bild 2.19 Bus Controller 8288 in System Bus Mode



2.17



2.19



Mikroprozessorsystem K 1810 WM86

Hardware · Software · Applikation (Teil 2)

Prof. Dr. Bernd-Georg Münzer (wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann, Wolfgang Kabatzke, Frank Kamrad, Helfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock, Sektion Technische Elektronik, Wissenschaftsbereich Mikrorechentechnik/Schaltungstechnik

3. Interface-Schaltkreise

Im 16-Bit-Mikroprozessorsystem 8086 werden folgende programmierbare Interface-Schaltkreise als parallele, serielle und Timer-Ports verwendet:

8253 Programmierbarer Zähler/Zeitgeberschaltkreis, PIT
8251A Programmierbarer serieller Interfaceschaltkreis, USART
8255A Programmierbarer paralleler Interfaceschaltkreis, PPI

Diese Schaltkreise wurden nicht speziell für das 16-Bit-System 8086 entwickelt, sondern aus dem 8-Bit-System 8080 übernommen und bezüglich der dynamischen Parameter weiterentwickelt. Bei den Schaltkreisen USART und PPI sind in 16-Bit-Systemen nur die A-Typen anzuwenden.

3.1. Programmierbarer Zähler-/Zeitgeberschaltkreis 8253 (PIT)

Der PIT-Schaltkreis (Programmable Interval Timer) realisiert im Mikroprozessorsystem 8086 die Zähl- und Zeitgeberfunktionen und weist folgende wesentliche Leistungsmerkmale auf:

- 3 unabhängige 16-Bit-Zähler mit Zähleingang, Zählausgang und Gatesteuerung
- 6 programmierbare Betriebsarten
- Zählen im Binär- oder BCD-Format
- max. Zählfrequenz von 2,0 MHz.

3.1.1. Architektur

Die Funktionseinheiten des PIT (Bild 3.1) haben folgende Aufgaben:

● **Datenbuspuffer**
Der bidirektionale/tristate 8-Bit-Puffer stellt die Schnittstelle zum Systemdatenbus dar. Das Senden oder Empfangen von Daten erfolgt mit E/A-Operationen der CPU. Über den Datenbuspuffer werden drei Basisfunktionen des PIT ausgeführt:
– Programmieren der Modes
– Laden des Zählregisters
– Lesen des Zählerstandes.

● **Lese-/Schreib-Logik**
Die Lese-/Schreib-Logik verarbeitet die Informationen der Eingänge Read $\overline{RD}$, Write $\overline{WR}$, Chip Select $\overline{CS}$ sowie die Portadressen A0 und A1 des Systemsteuerbusses. Die Dekodierung für die Portadressen und deren Zuordnung zu den internen Funktionseinheiten ist in Tafel 3.1 dargestellt.

● **Steuerwortregister**
Das Steuerwortregister (A0 = A1 = 1) verarbeitet die Information des Datenbuspuffers als Steuerwort zur Grundinitialisierung des PIT. Das Steuerwortregister kann nur beschrieben werden. Über OUT-Befehle der CPU zum Steuerwortregister erfolgt die unabhängige Modeeinstellung jedes Zählers.

● **Zähler 0–2**
Der Aufbau aller drei Zähler ist identisch. Jeder Zähler besteht aus einem voreinstellbaren 16-Bit-Rückwärtszähler ohne Vorteiler zum Zähltakt CLK. Das Zählen kann im Binär- oder BCD-Format erfolgen. Zähler-Eingang CLK, Toreingang GATE und Ausgang OUT sind entsprechend der Modeselektion verknüpft. Mit dem Toreingang GATE können für jeden Kanal die Zählvorgänge getriggert, gestartet und gestoppt werden. Alle Zähler arbeiten unabhängig und beeinflussen sich gegenseitig nicht. Auch die Modeeinstellung und das Laden von Zählwerten erfolgt für jeden Zähler getrennt. Das Lesen der aktuellen Zählerinhalte wird durch CPU-IN-Operationen auch während des Lesevorgangs realisiert.

3.1.2. Pinbelegung

Der PIT 8253 wird in einem 24-poligen Standard-DIL-Gehäuse gefertigt (Bild 3.2). Die Anschlüsse haben folgende ***Funktionen:***

D7–D0	8-Bit-Datenbus, bidirektional, 3-state
CLK 0–2	Zähler-Takteingänge
GATE 0–2	Toreingänge
OUT 0–2	Zählerausgänge
$\overline{RD}$	Read Lesesignal, Eingang low-aktiv
$\overline{WR}$	Write Schreibsignal, Eingang low-aktiv
$\overline{CS}$	Chip Select Steuersignal zur Bausteinauswahl, Eingang low-aktiv. Der Anschluß wird über einen Dekoder 8205 zum System-Adreßbus geführt.
A0, A1	Adreßeingänge für die Auswahl der Zähler 0, 1, 2 bzw. Steuerwortregister (Tafel 3.1). Diese Anschlüsse sind mit den Adressen AB1 und AB2 des Systems zu verbinden.
Vcc	Betriebsspannung + 5 V
GND	Masseanschluß

3.1.3. Betriebsarten

Die Zähler 0, 1, 2 des 8253 werden individuell

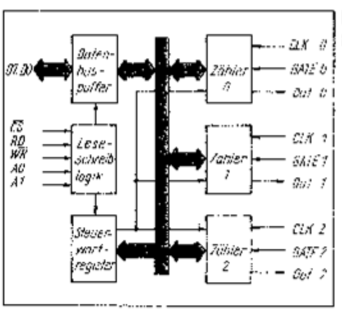


Bild 3.1 Architektur 8253-PIT

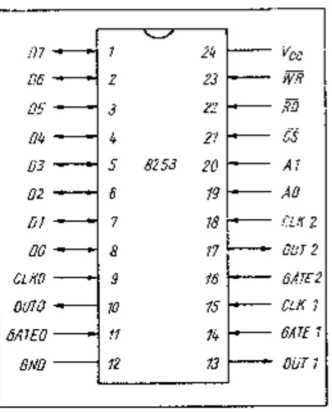


Bild 3.2 Anschlußbelegung 8253-PIT

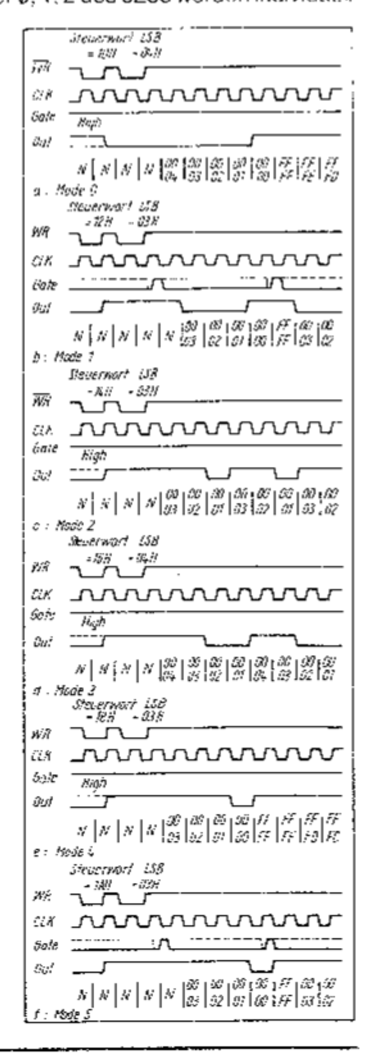
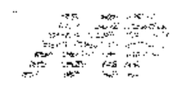


Bild 3.3 Betriebsarten des 8253-PIT
Bild 3.3a Mode 0,
Bild 3.3b Mode 1,
Bild 3.3c Mode 2,
Bild 3.3d Mode 3,
Bild 3.3e Mode 4,
Bild 3.3f Mode 5,

Tafel 3.1 ***8253 Portadressen***

A1	A0	Port
0	0	Zähler 0
0	1	Zähler 1
1	0	Zähler 2
1	1	Control (nur Schreiben)



durch Schreiben eines Steuerwortes in das Steuerwortregister initialisiert. Grundsätzlich stehen sechs verschiedene Betriebsarten zur Auswahl:

– ***MODE 0 = Interrupt on Terminal Count***
Nach dem Setzen des Modes geht OUT auf Low (Bild 3.3a). Nachdem die Zählkonstante geladen wurde, erfolgt ein Dekrementieren mit CLK. Beim Erreichen des Zählernullwertes geht OUT auf High und wird erst nach dem Laden einer neuen Zählkonstante oder nach einer neuen Modeeinstellung zurückgesetzt. Ein Low-Signal an GATE sperrt den Zählvorgang.

– ***MODE 1 = Programmable ONE-Shot***
Mit Mode 1 wird eine Monoflopfunktion mit Triggerung am GATE-Eingang realisiert (Bild 3.3b). Eine L/H-Flanke an GATE triggert den Zählvorgang, und OUT geht nach dem nächsten Eingangstaktimpuls auf Low. Beim erreichen des Zählernullwertes wird OUT konstant High. Eine weitere Triggerflanke wiederholt den Vorgang. Eine Triggerflanke bereits vor dem Erreichen des Zählernullwertes lädt den Rückwärtszähler neu und bewirkt wieder eine volle Auszählung ab der Zeit der Triggerung. Wird innerhalb des Dekrementierens ein neuer Zählerwert geladen, so wird zunächst noch der bisherige Zählerwert abgearbeitet, bevor dieser geladene Zählerwert mit erneuter Triggerung zur Wirkung kommt.

– ***MODE 2 = Rate-Generator***
Während des Dekrementierens des Rückwärtszählers verbleibt OUT = High (Bild 3.3c). Bei erreichen des Zählernullwertes wird OUT für eine Periodendauer von CLK gleich Low, und der Vorgang des Rückwärtszählens wiederholt sich in der nächsten Periode. Low-Signal an Gate sperrt den Zählvorgang. Ein Neuladen des Zählregisters während des Dekrementiervorgangs wird erst in der nächsten Periode wirksam. Mode 2 des PIT entspricht etwa der Zeitgeberfunktion des U 857-CTC.

– ***MODE 3 = Square Wave Rate-Generator***
Dieser Mode realisiert die Funktion eines Rechteckwellengenerators und ist im wesentlichen mit dem Mode 2 vergleichbar. Unterschiedlich ist hier das Tastverhältnis der OUT-Signale. Nach dem Laden eines geraden Wertes in das Zählregisters geht OUT für N/2 Taktimpulse zunächst auf High und für die weiteren N/2 Taktimpulse auf Low (Bild 3.3d). Der Rückwärtszähler wird beim Erreichen des Nullwertes automatisch wieder geladen, und der Vorgang wiederholt sich. Ein Low-Signal an GATE sperrt den Zählvorgang. Wenn der Wert im Zählregister ungerade ist, so wird OUT für (N+1)/2 Taktimpulse gleich High und für die folgenden (N–1)/2 Taktimpulse gleich Low. Die Betriebsart Mode 3 wird zur Generierung von Sende- bzw. Empfangstakt für den seriellen Interface-Schaltkreis USART-8251 A verwendet.

– ***MODE 4 = Softwaretriggered Strobe***
Nach dem Setzen des Modes und Laden der Zeitkonstante beginnt der Zählvorgang, und OUT verbleibt auf HIGH (Bild 3.3e). Beim Erreichen des Zählernullwertes wird OUT für eine Taktperiode gleich Low und verbleibt anschließend auf High. Erst nach dem erneuten Laden des Zählregisters wird dieser Vorgang wiederholt. Low-Signal an GATE sperrt den Zählvorgang.

– ***MODE 5 = Hardwaretriggered Strobe***
Diese Betriebsart ähnelt Mode 4, nur daß der Zählvorgang durch eine L/H-Flanke an Gate gestartet wird (Bild 3.3f).

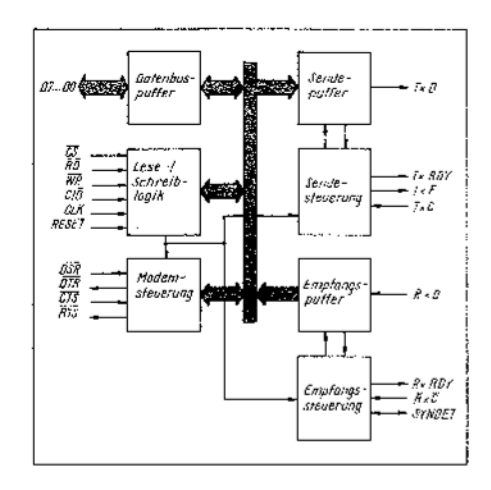
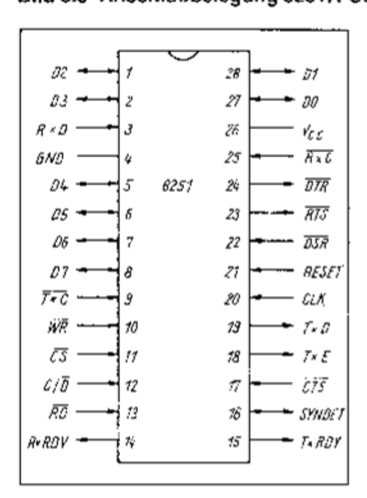


Bild 3.4 Architektur 8251-USART

Bild 3.5 Anschlußbelegung 8251A-USART



Tafel 3.2 8253 Aufbau des Steuerwortes

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

SC1	SC0	Zählerauswahl (Select Counter)
0	0	Zähler 0
0	1	Zähler 1
1	0	Zähler 2
1	1	unerlaubt

RL1	RL0	Lese- oder Ladeoperation (Read/Load)
0	0	Zähler-Latch-Operation (D3 ... D0 beliebig) Lesen/Laden der Zählerkonstante
0	1	Lesen/Laden des niederwertigen Bytes
1	0	Lesen/Laden des höherwertigen Bytes
1	1	Lesen/Laden des niederwertigen und danach des höherwertigen Bytes

M2	M1	M0	Modeauswahl
0	0	0	MODE 0
0	0	1	MODE 1
X	1	0	MODE 2
X	1	1	MODE 3
1	0	0	MODE 4
1	0	1	MODE 5

BCD	Zählformat
0	16-Bit, binär
1	4 Dekaden BCD

3.1.4. Programmierung

In der Grundinitialisierung des Systems wird jeder PIT-Zähler einzeln programmiert.
Im Steuerwort (Tafel 3.2) erfolgt mit D7, D6 die Selektion des gewünschten Zählers. Die Bits D3, D2, D1 legen die Betriebsart fest. Mit den Bits D5, D4 wird entweder eine 1 Byte Zählkonstante oder eine 2 Byte Zählkonstante festgelegt. Das Bit D0 bestimmt das Zählformat. Mit einem OUT-Befehl wird das Steuerwort zur Portadresse Control (A1 = A0 = 1) geschrieben. Danach erfolgt das Laden der Zählkonstante als 1-Byte- oder 2-Byte-Information zur Adresse des im Steuerwort selektierten Zählers mit einer Portadresse nach Tafel 3.1. Somit ergibt sich folgende Sequenz für die Programmierung eines Zählers N:

1. Steuerwort zum Control-Port
2. LSB Zählregister-Byte zum Zähler-N-Port
3. MSB Zählregister-Byte zum Zähler-N-Port

Nach Ausgabe des Steuerwortes können zu beliebigen Zeitpunkten die Zählregister der einzelnen Zähler neu beschrieben werden.
Der aktuelle Zählerstand wird durch IN-Operationen vom Zähler-Port bestimmt.

3.2. Programmierbarer serieller Interfaceschaltkreis 8251 A (USART)

Der USART-Schaltkreis (Universal Synchronous/Asynchronous Receiver/Transmitter) realisiert im Mikrocomputersystem 8086 die serielle Datenübertragung und besitzt folgende wesentliche Leistungsmerkmale:

- synchrone und asynchrone Übertragung
- doppelt gepufferter Sender und Empfänger
- Übertragung von Zeichen im Format von 5 bis 8 Bits
- Übertragungsgeschwindigkeit bis zu 64 kBaud
- Synchron-Mode:
  – automatische Sync-Einfügung
  – interne oder externe Zeichensynchronisation
- Asynchron-Mode:
  – Clockrate: × 1, × 16, × 64
- automatische Breakerkennung
- automatische Fehlererkennung

3.2.1. Architektur

Die USART-Funktionseinheiten (Bild 3.4) haben folgende Aufgaben:
Über den Datenbuspuffer, als Schnittstelle zum Systemdatenbus, werden Steuerworte/Daten transferiert:

– Schreiben des Modesteuerwortes
– Schreiben des Befehlssteuerwortes
– Schreiben der zu sendenden Daten
– Lesen der zu empfangenden Daten
– Lesen der Statusinformation

Die Lese/Schreiblogik leitet aus den Signalen Chip-Select $\overline{CS}$, Clock CLK, Read $\overline{RD}$, Write $\overline{WR}$, Control/Data C/$\overline{D}$ und RESET Steuersignale für die Schaltkreisfunktionen ab.

Der Sendepuffer erhält die Daten im parallelen Format vom Datenbuspuffer und konvertiert diese in einen seriellen Datenstrom, der zur Ausgabe geführt wird. Entsprechend dem Übertragungsmodus werden zusätzliche Zeichen oder Bits für Kontrollfunktionen eingefügt.
Der Empfangspuffer empfängt einen seriellen Datenstrom und wandelt diesen in ein paralleles Format um. Dabei werden die eingefügten Kontroll-Zeichen/Bits ausgewertet.

3.2.2. Pinbelegung

Der USART 8251A im 28-Pin-DIL-Gehäuse (Bild 3.5) hat die Anschlußbelegung mit folgenden ***Funktionen:***

D7–D0 8-Bit-Datenbus bidirektional, 3-state

C/$\overline{D}$ Control/Data, Eingang
Steuerleitung zur Selektion von Steuer- oder Datenworten
C/$\overline{D}$ = High Control/Status
C/$\overline{D}$ = Low Data

$\overline{RD}$ Read

$\overline{WR}$ Write

$\overline{CS}$ Chip-Select

Diese Funktionen sind mit denen des 8253-PIT identisch.

CLK Systemtakt, Eingang
Der USART-Systemtakt (max. 3 MHz) ist mit dem Peripherietakt PCLK des Clockgenerators 8284A zu verbinden.

RESET Rücksetzen, Eingang high-aktiv

$\overline{TxC}$ Sendertakt (Transmitter Clock), Eingang
max. Frequenz:
1 × Bitrate 64 kHz
16 × Bitrate 310 kHz
64 × Bitrate 615 kHz

TxD serielle Sendedaten (Transmitter Data), Ausgang

$\overline{RxC}$ Empfängertakt (Receiver Clock), Eingang
max. Frequenz: vgl. **$\overline{TxC}$**

RxD serielle Empfangsdaten (Receiver Data), Eingang

TxRDY Sender bereit (Transmitter Ready), Ausgang
Dieser Ausgang liefert der CPU die Information, daß der USART zur Datenübernahme in Form eines OUT-Befehls bereit ist. Beim Schreiben eines Zeichens in den USART erfolgt ein automatisches Rücksetzen von TxRDY. TxRDY kann zur Interrupt-Auslösung am Interrupt-Controller 8259A verwendet werden, um in der Interrupt-Service-Routine eine Daten-Schreiboperation vorzunehmen.

RxRDY Empfänger bereit (Receiver Ready), Ausgang
Dieser Ausgang liefert der CPU die Information, daß der USART-Empfangspuffer ein Zeichen enthält und damit die Bereitschaft zur Übertragung in die CPU in Form eines IN-Befehls vorliegt. RxRDY wird mit $\overline{RD}$ zurückgesetzt. RxRDY kann über den Interrupt-Controller eine Interrupt-Service-Routine zum USART-Lesen einleiten.

$\overline{DSR}$ Data Set Ready, Eingang, low-aktiv
Steuerleitung für den Modembetrieb
$\overline{DSR}$ kann im Bit 7 des Statusregisters gelesen werden.
$\overline{DSR}$ hat keinen Einfluß auf das Senden/Empfangen von Daten.

$\overline{DTR}$ Data Terminal Ready, Ausgang low-aktiv
$\overline{DTR}$ wird durch Bit 1 im Befehlssteuerwort gesetzt/rückgesetzt.

$\overline{RTS}$ Request To Send, Ausgang low-aktiv
Steuerleitung für den Modembetrieb
$\overline{RTS}$ wird durch Bit 5 im Befehlssteuerwort gesetzt/rückgesetzt.

$\overline{CTS}$ Clear To Send, Eingang, low-aktiv
$\overline{CTS}$ = Low startet das Senden der seriellen Daten an TxD, wenn das Bit 0 im Befehlssteuerwort (Sendefreigabe) gleich „1" gesetzt wurde. Ein Software-Start des Sendens ohne $\overline{CTS}$ = Low ist beim USART nicht möglich.

TxE Sendepuffer leer (Transmitter Empty), Ausgang, high-aktiv
TxE liefert High-Signal, wenn der Parallel-Serien-Converter im Transmitter leer ist. TxE wird automatisch gleich Low, wenn ein Byte in den Sendepuffer geschrieben wurde.

SYNDET Ein-/Ausgang
Synchronisations- oder Breakerkennung
Synchron-Mode (SYNC-Detect)
In dem internen Synchronisationsmode notiert der Ausgang SYNDET gleich High ein gefundenes SYNC-Zeichen. Im Bisync-Mode geht der Ausgang SYNDET bei der Erkennung des letzten Bits des zweiten SYNC-Zeichens auf High. Bei einer Status-Leseoperation wird SYNDET automatisch zurückgesetzt. Wenn externer SYNC-Mode programmiert ist, wirkt SYNDET als Eingang, und ein positiver Pegelübergang an SYNDET startet die Zeichenerfassung am seriellen Dateneingang RxD.
Asynchron-Mode (Break Detect)
Ausgang
High-Signal wird ausgegeben, wenn alle Zeichen des seriellen Datenprotokolls (Start-Bit, Daten-Bit, Paritätsbit, 1 Stop-Bit) als 0 erkannt werden.

Vcc Betriebsspannung + 5 V

GND Masseanschluß

Tafel 3.3 8251A Steuersignale (mit $\overline{CS}$ = 0)

$\overline{RD}$	$\overline{WR}$	C/$\overline{D}$	Funktion
0	1	0	Daten vom USART zum Systembus (Portlesen)
0	1	1	Statuslesen vom USART
1	0	0	Daten vom Systembus zum USART (Portschreiben)
1	0	1	USART-Steuerwort schreiben

Tafel 3.4 8251A Aufbau des Modesteuerwortes

Asynchron-Mode

D7	D6	D5	D4	D3	D2	D1	D0
S2	S1	EP	PEN	L2	L1	B2	B1

S2	S1	Anzahl der Stopbits
0	0	nicht erlaubt
0	1	1 Bit
1	0	1 1/2 Bits
1	1	2 Bits

EP	Paritätsgenerierung/-test (Even Parity Generation/Check)
0	ungerade
1	gerade

PEN	Paritätsfreigabe (Parity enable)
0	gesperrt
1	freigegeben

L2	L1	Zeichenlänge
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

B2	B1	Bitratefaktor
0	0	nicht erlaubt (Synchron-Mode)
0	1	×1
1	0	×16
1	1	×64

Synchron-Mode

D7	D6	D5	D4	D3	D2	D1	D0
SCS	ESD	EP	PEN	L2	L1	0	0

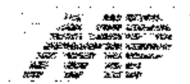
SCS	Anzahl der SYNC-Zeichen Single-Character Sync
0	zwei Sync-Zeichen
1	ein Sync-Zeichen

ESD	externe Synchronisationserkennung (External Sync Detect)
0	Intern-Sync
1	Extern-Sync

EP	Paritätsgenerierung/-test (Even Parity Generation/Check)
0	ungerade
1	gerade

PEN	Paritätsfreigabe (Parity enable)
0	gesperrt
1	freigegeben

L2	L1	Zeichenlänge
0	0	[illegible]
0	1	[illegible]
1	0	7 Bits
1	1	[illegible] Bits



3.2.3. Programmierung

Die Funktion des USART wird durch Programmierung mit zwei Steuerwörtern, dem Modesteuerwort und dem Befehlssteuerwort, festgelegt. Im Modesteuerwort (Tafel 3.4) werden folgende Vereinbarungen getroffen:
- Synchron- oder Asynchron-Mode
- Bitratenfaktor im Asynchron-Mode (x1, x16, x64)
- Zeichenlänge (5 ... 8 Bit)
- Paritätskontrolle
- Anzahl der Stopbits (1, $1^1/_2$, 2)
- Synchronisations-Steuerung (intern, extern, Double SYNC-Character, Single SYNC-Character)

Das Befehlssteuerwort (Tafel 3.5) legt folgende Parameter fest:
- Sender-/Empfängerfreigabe
- Setzen der Modemsteuersignale DTR, RTS
- Reset-Funktionen

Die Programmierung von Modesteuerwort und Befehlssteuerwort muß in einer definierten Reihenfolge vorgenommen werden. Nach dem internen oder externen Rücksetzen des 8251A wird das erste Steuerwort als Modesteuerwort interpretiert, alle folgenden Steuerwörter werden als Befehlssteuerwörter erkannt. Das interne Rücksetzen kann im Befehlssteuerwort (Bit 6 = 1) festgelegt werden. Die Betriebsarten des USART entsprechen im wesentlichen den bekannten Prinzipien für Asynchron-/Synchron-Mode. Im SYNC-Mode besitzt der USART eine geringere Leistungfähigkeit als der Schaltkreis U 856-SIO.

Tafel 3.5 8251A Aufbau des Befehlssteuerwortes

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RxEN	DTR	TxEN

EH	Suchen von Sync-Zeichen (Error Hunt) (nur im Synchron-Mode)
0	Suchbetrieb stop
1	Suchbetrieb freigegeben
IR	internes RESET
0	Verbleiben im Befehlssteuerwort
1	Rückkehr zum Modesteuerwort
RTS	Sendeanforderung (Request To Send)
0	Pin $\overline{RTS}$ – High (inaktiv)
1	Pin $\overline{RTS}$ – Low (aktiv)
ER	Fehler-Reset (Error Reset)
0	keine Beeinflussung der Fehler-Flags
1	Rücksetzen der Fehlerflags PE, OE und FE
SBRK	Sendeunterbrechung (Send Break Character)
0	normale Operation
1	PIN TxD – Low
RxEN	Empfängerfreigabe (Receive Enable)
0	Empfang gesperrt
1	Empfänger freigegeben
DTR	Datenendstelle bereit (Data Terminal Ready)
0	Pin $\overline{DTR}$ = High (inaktiv)
1	Pin $\overline{DTR}$ – Low (aktiv)
TxEN	Senderfreigabe (Transmit Enable)
0	Senden gesperrt
1	Sender freigegeben

● **Asynchron-Mode**

Das Datenformat ist im Bild 3.6 dargestellt.

Senden

Im passiven Zustand liegt das High-Signal am Ausgang TxD. Das Senden beginnt bei $\overline{CTS}$ – Low mit einem Startbit, dann folgen die Datenbits ab D0 bis zur programmierten Zeichenlänge. Den Daten folgt ein Paritätsbit, falls dieses im Modesteuerwort freigegeben wurde. Den Abschluß bildet die programmierte Anzahl der Stopbits. Wenn der Zeichenpuffer leer ist und keine Breakausgabe im Befehlssteuerwort programmiert wurde, geht TxD auf High. Das Ende des Sendens wird mit einem High-Pegel am Pin TxRDY bzw. im Statusbit notiert. Die seriellen Daten werden mit der fallenden Flanke von $\overline{TxC}$, geteilt durch den programmierten Bitratefaktor, gesendet.

Empfangen

Ein konstanter High-Pegel an RxD wird als Ruhezustand interpretiert. Eine fallende Flanke an RxD kennzeichnet den Beginn des Startbits. Der Pegel an RxD wird mit der steigenden Flanke von $\overline{RxC}$, geteilt durch den Bitratefaktor, abgefragt und Datenbits, Paritätsbit (falls programmiert) und Stopbits entsprechend Bild 3.4 in einen Serien-Parallel-Wandler übernommen. Beträgt die Zeichenlänge weniger als 8 Bits, so werden für die nicht vorhandenen Bits Nullen eingefügt. Bei einem Paritätsfehler wird das Parity-Error-Flag gesetzt. Falls nach dem Paritätsbit ein Low-Pegel als Stopbit festgestellt wird, so erfolgt ein Setzen des Framing-Error-Flags. Unabhängig von der Anzahl der programmierten Stopbits fordert der Empfänger nur ein Stopbit. Ist der Datenbuspuffer geladen, so wird das Pin bzw. Statusbit RxRDY gesetzt. Damit wird der CPU mitgeteilt, daß ein Byte zur Abholung in Form eines IN-Befehls bereitsteht. Wenn dieses von der CPU nicht eingelesen wurde, so erfolgt ein Überschreiben mit dem nächsten empfangenen Zeichen, dabei wird das Overrun-Error-Flag gesetzt.

● **Synchron-Mode**

Das Datenformat ist im Bild 3.7 dargestellt.

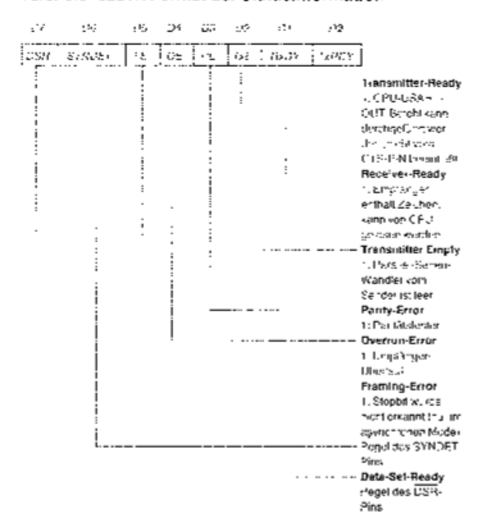
Senden

Im Synchron-Mode sind die Daten von SYNC-Zeichen eingeschlossen. Mit $\overline{CTS}$ – Low werden die Daten mit der fallenden Flanke von $\overline{TxC}$ an TxD hinausgeschoben. Nachdem der Sendepuffer leer ist, werden automatisch Sync-Zeichen in den Datenstrom eingefügt, bis ein neues Byte in den 8251A geschrieben wurde. Der Ausgang Transmitter Empty TxE = High teilt der CPU mit, daß der Sendepuffer leer ist, und TxE wird erst mit dem nächsten OUT-Befehl auf Low zurückgesetzt.

Empfangen

Die Zeichensynchronisation kann intern oder extern ausgeführt werden. Im Befehlssteuerwort sollte der Suchbetrieb mit D7 = 1 freigegeben werden. Mit der steigenden Flanke von $\overline{RxC}$ werden die an RxD liegenden Pegel

Tafel 3.6 8251A Format der Statusinformation



Tafel 3.7 Pollinggesteuerte Ein-/Ausgabe für USART

```
INPUT:  IN   AL,USART_CONTROL  ;Lesen Statusregister
        TEST AL,2              ;Prüfen RxDY = 1
        JZ   INPUT
        IN   AL,USART_DATA     ;gültiges Zeichen
                               ;vorhanden, Daten lesen
        PUSH AX                ;in AL auszugebendes
                               Byte
OUTPUT: IN   AL,USART_CONTROL  ;Lesen Statusregister
        TEST AL,1              ;prüfen TxRDY = 1
        JZ   OUTPUT
        POP  AX                ;Sendepuffer leer!
        OUT  USART_DATA,AL     ;Daten schreiben
```

übernommen. Wenn die Sync-Zeichen erkannt wurden, beendet der USART den Suchbetrieb und ist synchronisiert. Der Anschluß SYNDET wird anschließend auf High gesetzt und beim Statuslesen automatisch zurückgesetzt. Im externen Synchronisationsmode wird die Synchronisation durch den High-Übergang am Eingang SYNDET bis zum nächsten $\overline{RxC}$-Zyklus gestartet. Paritäts- und Überlauffehler werden wie beim asynchronen Empfang überprüft.

3.2.4. Statusregister

Das Statusregister enthält Informationen über die Sende-/Empfangs- und Fehlerbedingungen und den logischen Pegel einiger Kommunikationssignale (Tafel 3.6). Das Statusregister kann durch IN-Befehl von der USART-Control-Adresse (C/$\overline{D}$ = 1) gelesen werden.

Die Statusbits $\overline{DSR}$, SYNDET, TxE und RxRDY geben den aktuellen Logikpegel der entsprechenden Anschlußpins wieder. TxRDY informiert, daß ein neues Zeichen von der CPU in den Datenbuspuffer geschrieben werden kann und ist unabhängig von einer programmierten Sendefreigabe (TxEN-Bit im Befehlssteuerwort) und vom $\overline{CTS}$-Pin. Die Statusbits

FE Framing Error: Stopbit-Fehler,

OE Overrun Error: Zeichen wurde von der CPU nicht abgeholt,

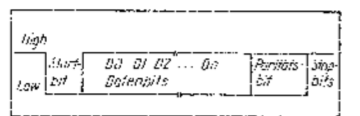


Bild 3.6 Datenformat im Asynchron-Mode

Bild 3.7 Datenformat im Synchronmode

Sync-Zeichen 1	Sync-Zeichen 2	Daten (5-8 Bits/Zeichen)	Daten

PE Paritätsfehler
dienen als Fehlerflags.
Für die Polling-gesteuerte Eingabe/Ausgabe werden die Statusbits RxRDY und TxRDY verwendet (Tafel 3.7).

3.3. Programmierbarer paralleler Interfaceschaltkreis 8255A (PPI)

Der PPI-Schaltkreis 8255A (Programmable Peripheral Interface) realisiert im Mikroprozessorsystem 8086 die parallele Ein-/Ausgabe und ist durch folgende Leistungsmerkmale gekennzeichnet:
- 3 programmierbare Ein-/Ausgabeports; Port A, B, C
- 3 Betriebsarten
Mode 0: Basic Input/Output
Mode 1: Strobed Input/Output
Mode 2: Strobed Bidirectional Bus
- Einzelbit-Set/Reset-Operation an Port C
- Interruptauslösung in Mode 1 und 2 in Verbindung mit Interrupt-Controler 8259A
- kein Systemtakt erforderlich

3.3.1. Architektur

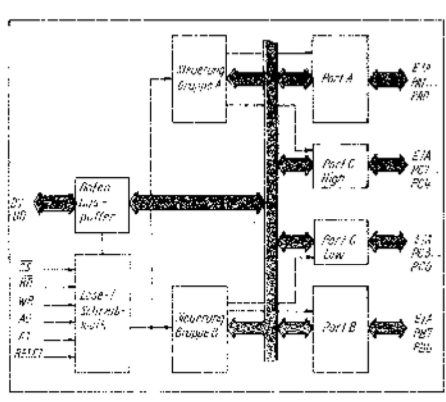
Die Architektur des PPI zeigt Bild 3.8. Die Funktionen des Datenbuspuffers und der Lese-/Schreiblogik entsprechen denen des 8253 PIT. Die Adressen für Port A, B, C und für das Steuerwort sind in Tafel 3.8 dargestellt. Auf der Peripherieseite werden die Ports in zwei Gruppen eingeteilt:
Gruppe A: 8-Bit-Port A PA7 – PA0
4-Bit-Port C PC7 – PC4
Gruppe B: 8-Bit-Port B PB7 – PB0
4-Bit-Port C PC3 – PC0
Die Programmierung mit Modeeinstellung erfolgt für beide Gruppen zusammen in einem Steuerwort.
Die Ports haben folgende Eigenschaften:

Port A
Das Port A wird vorzugsweise als 8-Bit-Ein-/Ausgabe-Latch/Puffer verwendet. Der bidirektionale Betrieb ist nur mit Port A möglich.

Port B
Port B wird vorzugsweise als 8-Bit-Ausgangs-Latch/Puffer oder als 8-Bit-Eingangslatch verwendet.

Port C
Port C wird in zwei 4-Bit-Ports aufgeteilt:
PC7 ... PC4
PC3 ... PC0
Die Teilports können einzeln als Ausgangs-Latch/Puffer oder Eingangspuffer in Mode 0 programmiert werden. In den Handshaking-

Bild 3.8 Architektur 8255A-PPI



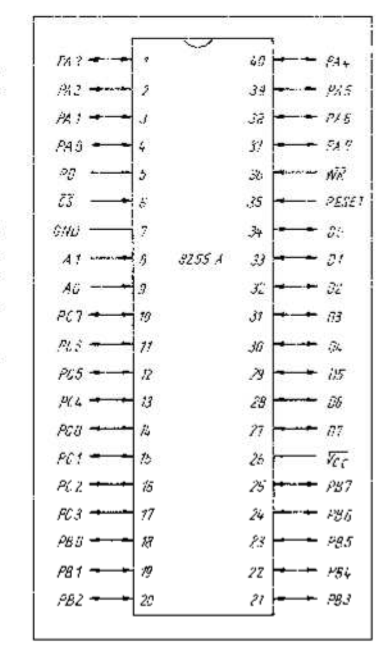


Bild 3.9 Anschlußbelegung 8255A-PPI

Tafel 3.8 8255A Portadressen

A1	A0	Port
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Steuerwort (nur Schreiben)

Tafel 3.9 8255A Aufbau des Steuerwortes

	D7	D6	D5	D4	D3	D2	D1	D0
		Gruppe A				Gruppe B		
Modesteuerung →	S=1	M2	M1	A	CH	M0	B	CL
Einzel-Bit-Ausgabe Port C →	S=0	X	X	X	C2	C1	C0	S/R

Modesteuerung (S = 1)

M2	M1	Modeauswahl
0	0	Mode 0
0	1	Mode 1
1	X	Mode 2

A Auswahl Port A, PA7 ... PA0 ╲ als Ausgang: 0
CH Auswahl Port-C-High PC7 ... PC4 ╱ oder Eingang: 1

M0	Modeauswahl
0	Mode 0
1	Mode 1

B Auswahl Port B, PB7 ... PB0 ╲ als Ausgang: 0
CL Auswahl Port-C-Low PC3 ... PC0 ╱ oder Eingang: 1

Einzel-Bit-Ausgabe für Port C (S = 0)

C2	C1	C0	Bitauswahl für Port C
0	0	0	Portleitung PC0
0	0	1	Portleitung PC1
0	1	0	Portleitung PC2
0	1	1	Portleitung PC3
1	0	0	Portleitung PC4
1	0	1	Portleitung PC5
1	1	0	Portleitung PC6
1	1	1	Portleitung PC7

S/R	Setz-/Rücksetzfunktion für PC0 ... PC7
0	rücksetzen
1	setzen

Betriebsarten werden einige Port-C-Leitungen als Steuersignale den Ports A und B zugeordnet.

3.3.2. Pinbelegung

Der PPI 8255A wird in einem 40-poligen DIL-Gehäuse gefertigt. Die Anschlüsse (Bild 3.9) haben folgende Funktionen:

D7–D0 8-Bit-Datenbus bidirektional, 3-state
$\overline{RD}$ $\overline{WR}$ $\overline{CS}$ entsprechen den Funktionen des 8253-PIT.
RESET Rücksetzen, Eingang high-aktiv Löscht das Steuerwortregister, und alle Portanschlüsse werden als Eingang geschaltet.
A0 A1 Adreßeingänge zur Portauswahl Diese Signalleitungen adressieren die E/A-Ports und das Steuerwortregister nach Tafel 3.8. A0 und A1 werden zum 8086-System-Adreßbus AB1 und AB2 geführt.
PA0–PA7 Port A
$U_{OL} \leqq 0{,}45$ V, $I_{OL} \leqq 1{,}7$ mA
$U_{OH} \geqq 2{,}40$ V, $I_{OH} \geqq -200\,\mu$A
PB0–PB7 Port B
PC0–PC7 Port C
Port B,C Darlington-Strom
$R_{ext} = 750$ Ohm
$U_{ext} = 1{,}5$ V
$I = -1{,}0 \ldots 4{,}0$ mA
Vcc Betriebsspannung –5 V
GND Masseanschluß

3.3.3. Programmierung

Die Programmierung des 8255A erfolgt mit einem Steuerwort (Tafel 3.9) in zwei Varianten.
Modesteuerwort (D7 = 1):
Programmierung der Ports A, B, C in den Betriebsarten 0, 1, 2
Einzel-Bit-Ausgabe (D7 = 0):
Set-/Reset-Ausgaben für jedes Bit von Port C einzeln.

Modesteuerwort
Die Modeeinstellung erfolgt getrennt für die Gruppen A und B, dabei ergeben sich für die Ports A, B, C folgende Varianten:
Port A: Mode 0 oder 1 oder 2
Port B: Mode 0 oder 1
Port C: Mode 0

– MODE 0 = Basic Input/Output
In Mode 0 werden Ein-/Ausgabe-Operationen ohne Quittung vorgenommen. Wenn alle Ports des 8255A in Mode 0 programmiert werden (M2 = M1 = M0 = 0), so stehen folgende Ein-/Ausgaben zur Verfügung:
- zwei 8-Bit-Ports für Byte Ein-/Ausgabe (Port A, B)
- zwei 4-Bit-Ports für Halbbyte-Ein-/Ausgabe (Port C)
Die Portausgänge sind gelatcht, die Porteingänge sind nicht gelatcht.

– MODE 1 = Strobed Input/Output
Die Ports A und B arbeiten als Ein- oder Ausgabeport im Quittungsbetrieb in Verbindung mit dem Port C, welches die Handshakingsignale verwaltet. In den quittungsgesteuerten Ein-/Ausgaben kann vom 8255A eine Interrupt-Anforderung an den Programmierbaren-Interrupt-Controller 8259A angemeldet werden. Die Interruptfreigabe des 8255A wird

durch ein internes INTE-Flip-Flop organisiert, welches durch Einzelbit-Set/Reset-Operationen von Port C beeinflußt wird.
Bit-Set – Interrupt-Freigabe
Bit-Reset = Interrupt-Sperre
Die Steuersignale haben im Mode 1 folgende ***Funktion:***

Steuersignale für Port-Input-Operationen

$\overline{STB}$ Strobe, Eingang, low-aktiv
$\overline{STB}$ – Low lädt die Daten in das Port-Eingangslatch.

IBF Input Buffer Full, Ausgang low-aktiv
High notiert, daß Daten in das Eingangslatch geladen worden sind und stellt somit ein Bestätigungssignal dar. IBF wird mit $\overline{STB}$ = Low gesetzt und mit der Rückflanke von $\overline{RD}$ zurückgesetzt.

INTR Interrupt Request, Ausgang, high-aktiv
INTR wird gesetzt, wenn nach dem Latchen der Port-Eingabedaten $\overline{STB}$ und IBF gleich High sind. In der daraufhin eingeleiteten Interrupt-Service-Routine mit Lesen der Port-Eingabe-Daten wird INTR mit der Vorderflanke von $\overline{RD}$ zurückgesetzt. Die INTE-Flip-Flops von Port A und B werden kontrolliert durch:
INTE Port A: Bit Set/Reset PC4
INTE Port B: Bit Set/Reset PC2

Steuersignale für Port-Output-Operationen

$\overline{OBF}$ Output Buffer Full, Ausgang low-aktiv
$\overline{OBF}$ aktiv = Low notiert, daß die CPU Daten in das Port geschrieben hat, die an den Portausgängen gültig bereitstehen. Diese Aktivierung von $\overline{OBF}$ erfolgt nach der Rückflanke von $\overline{WR}$.

$\overline{ACK}$ Acknowledge Input, Eingang, low-aktiv
$\overline{ACK}$ gleich Low notiert, daß die Peripherie vom 8255A die gültigen Daten übernommen hat. Mit $\overline{ACK}$ = Low wird $\overline{OBF}$ wieder inaktiv = High.

INTR Interrupt Request, Ausgang, high-aktiv
Mit $\overline{OBF}$ = High und $\overline{ACK}$ = High löst ein aktives INTR = High einen Interrupt aus, der in der Interrupt-Service-Routine zum Schreiben neuer Port-Ausgabe-Daten führt.
Mit der Vorderflanke von $\overline{WR}$ wird INTR inaktiv gleich Low, und am Ende des Bestätigungs-Zyklus mit der Rückflanke von $\overline{ACK}$ wird INTR wieder aktiv gleich High und damit ein neuer Interrupt ausgelöst.
Die INTE-Flip-Flops von Port A und Port B werden kontrolliert durch:
INTE Port A: Bit Set/Reset PC6
INTE Port B: Bit Set/Reset PC2

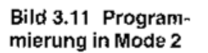
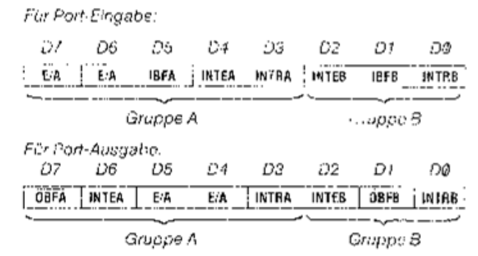


Bild 3.10 Varianten von Mode 1; Strobed Input/Output
Bild 3.10a Mode 1, Port A – Input
Bild 3.10b Mode 1, Port A – Output
Bild 3.10c Mode 1, Port B – Input
Bild 3.10d Mode 1, Port B – Output

Tafel 3.10 8255A Format der Statusinformation bei MODE 1 (IN-PORT C)



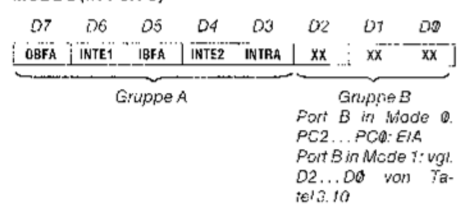
Für Port-Eingabe:

D7	D6	D5	D4	D3	D2	D1	D0
E/A	E/A	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB
Gruppe A					Gruppe B		

Für Port-Ausgabe:

D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTEA	E/A	E/A	INTRA	INTEB	OBFB	INTRB
Gruppe A					Gruppe B		

Tafel 3.11 8255A Format der Statusinformation bei MODE 2 (IN-Port C)



D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTE1	IBFA	INTE2	INTRA	XX	XX	XX
Gruppe A					Gruppe B		

Gruppe B: Port B in Mode 0: PC2 ... PC0: E/A; Port B in Mode 1: vgl. D2 ... D0 von Tafel 3.10

Die Varianten der Programmierung in Mode 1, getrennt nach Port A und B, jeweils für Eingang/Ausgang zeigen die Bilder 3.10a ... d. Informationen über den Zustand der Bestätigungssignale IBF, $\overline{OBF}$, des Interrupt-Freigabe-Flip-Flops INTE und der Interrupt-Anforderung INTR erhält man durch Lesen eines Status-Wortes von Port C (Vergl. Tafel 3.10).

– *MODE 2 = Strobed Bidirectional Bus I/O*
Die bidirektionale Port-Ein-/Ausgabe im Quittungsbetrieb wird nur über Port A realisiert. Am Port C befinden sich die Steuersignale für die Ausgabe $\overline{OBFA}$ und $\overline{ACKA}$ und für die Eingabe $\overline{STBA}$ und IBFA und das Interrupt-Anforderungssignal INTRA. Die Interrupt-Freigabe-Flip-Flops werden wiederum mit Bit-Setz-/Rücksetzfunktionen von Port C beeinflußt.
Ausgabe: Bit Set/Reset PC6 (INTE1)
Eingabe: Bit Set/Reset PC4 (INTE2)
Bild 3.11 zeigt den Aufbau des Modesteuerwortes, die zugehörige Pinbelegung von Port A und die Steuerleitungen an Port C. Wenn Mode 2 programmiert wurde, kann für Port B noch gewählt werden:
Mode 1: Strobed Input/Output mit den Handshaking-Signalen an PC0, PC1, PC2 nach Bild 3.10c und 3.10d
Mode 0: Basic Input/Output für PB0 ... PB7 mit frei wählbaren E/A-Leitungen an PC0 ... PC2.
Das Format der Statusinformation von Mode 2 ist in Tafel 3.11 dargestellt.

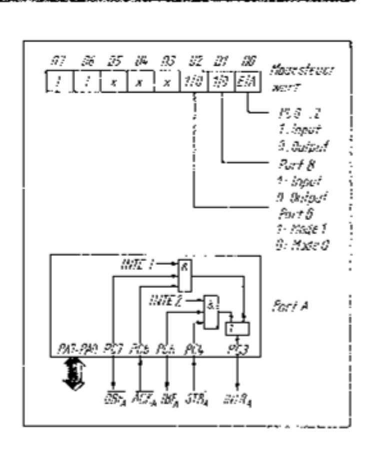


Bild 3.11 Programmierung in Mode 2

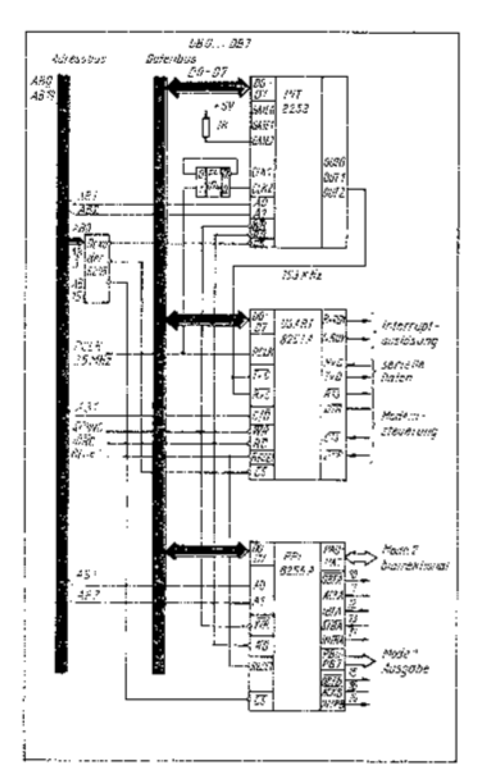


Bild 3.12 Interface-Schaltkreise im System 8086

Tafel 3.12 Initialisierung der Interfaceschaltkreise

```
;USART – INITIALISIERUNG
        MOV     DX,ØDAH      ;Portadresse USART, Control
        MOV     AL,ØØ        ;Vor-Reset
        OUT     DX,AL
        MUL     AL           ;Delay
        MUL     AL           ;
        OUT     DX,AL
        MUL     AL
        MUL     AL
        OUT     DX,AL
        MUL     AL
        MUL     AL
        MOV     AL,4ØH       ;Reset
        OUT     DX,AL
        MUL     AL
        MUL     AL
        MOV     AL,5AH       ;MODE-Steuerwort
                             ;Bitratefaktor 16, asynchron,
                             ;7 Bit/Charakter,
                             ;ungerade Parität frei-
                             ;gegeben, 1 Stopbit
        OUT     DX,AL
        MUL     AL
        MUL     AL
        MOV     AL,17H       ;Befehlssteuerwort
                             ;kein internes Reset,
                             ;/RTS inaktiv,
                             ;Rücksetzen der Fehlerflags,
                             ;Empfängerfreigabe,
                             ;/DTR aktiv, Senderfreigabe
        OUT     DX,AL
        MUL     AL
        MUL     AL
;PIT – INITIALISIERUNG
        MOV     DX,ØD6H      ;Portadresse PIT, Control
        MOV     AL,ØB6H      ;Zähler 2, 2 Byte-Zähler-
                             ;konstante,
                             ;MODE 3, 16 Bit-binaer
        OUT     DX,AL
        MOV     DX,ØD4H      ;Portadresse Zähler 2
        MOV     AL,08        ;Zählkonstante LSB
        OUT     DX,AL
        MOV     AL,ØØ        ;Zählkonstante MSB
        OUT     DX,AL        ;CLK: 1,23 MHz
                             ;OUT: 153 kHz
;PPI – INITIALISIERUNG
        MOV     DX,ØCEH      ;Portadresse PPI, Control
        MOV     AL,ØC4H      ;MODE-Steuerung
                             ;MODE 2 für Port A
                             ;MODE 1 für Port B, Portaus-
                             ;gabe:
                             ;(Bit 7, Bit 6, Bit 2 – 1;
                             ;Bit 1 = 0;
                             ;Bit 5, Bit 4, Bit 3, Bit 0:
                             ;beliebig = 0)
        OUT     DX,AL
        MOV     AL,Ø5        ;Interruptfreigabe für Port-
                             ;eingabe B
                             ;INTEB = PC2: set
        OUT     DX,AL
        MOV     AL,Ø9        ;Interruptfreigabe für Port-
                             ;ausgabe A
                             ;INTE2 = PC4: set
        MOV     DX,ØD8H      ;Portadresse USART, Data
                             ;Dateneingabe
        IN      AL,DX        ;≙ RxRDY Statusbit löschen
```

Einzel-Bit-Ausgabe

Jedes der 8 Bits von Port C kann einzeln gesetzt oder rückgesetzt werden durch Einzel-Bit-Ausgabe-Operationen von Port C nach Tafel 3.9 mit D7 = Ø. Wenn Port C in den Handshaking-Betriebsarten Mode 1 oder 2 als Status/Control-Wort für Port A oder B verwendet wird, kann ein gezieltes Freigeben/Sperren der Interrupt-Logik mit INTE-A, INTE-B (Tafel 3.10) und INTE-1, INTE-2 (Tafel 3.11) erfolgen. Die anderen Statusbits sind nicht durch Einzel-Bit-Ausgaben beeinflußbar.

3.4. Anwendungsbeispiel

In Bild 3.12 ist eine applikative Lösung für die Einbindung der Interfaceschaltkreise 8253, 8251A und 8255A in das Mikrorechnersystem 8086 dargestellt. Ein entsprechendes Programmierbeispiel für deren Initialisierung zeigt Tafel 3.12. Der Kanal 2 des PIT wird zur Erzeugung der Taktsignale für eine asynchrone Datenübertragung mit 9,6 kBaud und 7 Bit-ASCII-Zeichen genutzt. Bei der Taktversorgung des PIT 8253 ist darauf zu achten, daß die maximale Eingangsfrequenz von 2,0 MHz nicht überschritten wird und deshalb bei der Verwendung des Signals PCLK eine 2:1-Frequenzteilung vorgenommen werden muß. Der PPI ist für den Handshaking-Betrieb der Ports A und B initialisiert, wobei mit Port A ein bidirektionaler Betrieb und mit Port B eine Dateiausgabe in Mode 1 vorgesehen ist. Um eine sichere Initialisierung der Interface-Schaltkreise im System 8086 zu gewährleisten, müssen jeweils zwischen die Steueranweisungen zwei MUL AL-Befehle zur Verzögerung programmiert werden (beim USART zwingend erforderlich). Bei der softwaremäßigen RESET-Erzeugung für den USART wird durch die Ausgabe von Vor-Resets ein sicheres Rücksetzen gewährleistet.

4. Interruptsystem 8086

4.1. Interruptorganisation

Die CPU 8086 besitzt zwei Interrupteingänge für nichtmaskierbare (NMI) und maskierbare (INTR) Interrupts. Maskierbare Interrupts werden durch das CPU-Interrupt-Flag freigegeben oder gesperrt. Für 256 mögliche Interrupt-Service-Routinen sind die Startadressen mit Offset- und Segmentanteil in einer 1-KByte-Interrupt-Tabelle zu Beginn des Speicher-Adreßraumes nach Bild 4.1 plaziert. Die Interrupts werden mit dem Index der Adreßeintragung in der Interrupttabelle gekennzeichnet.

Die ersten fünf Interrupts sind nichtmaskierbar und CPU-internen Funktionen zugeordnet:

Interrupt Ø: von der CPU bei Division durch Null ausgelöst (divide error)

Interrupt 1: von der CPU nach jeder Befehlsausführung ausgelöst, falls T-Flag = 1 gesetzt ist (single step)

Interrupt 2: Nichtmaskierbarer Interrupt (NMI)

Interrupt 3: 1-Byte-Interrupt-Befehl INT3 (one byte interrupt)

Interrupt 4: von der CPU nach dem Befehl INTO ausgelöst, falls das Overflow-Flag gesetzt ist (on overflow)

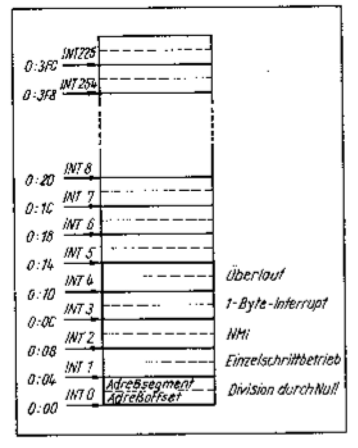


Bild 4.1 Interrupttabelle

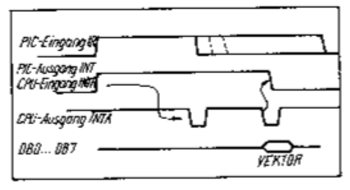


Bild 4.2 Signalverlauf zwischen PIC und CPU

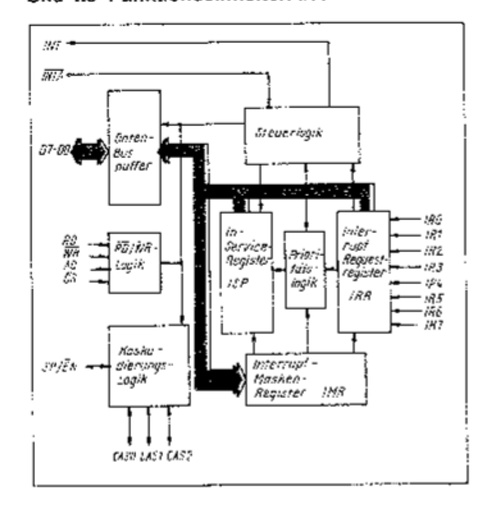


Bild 4.3 Funktionseinheiten des PIC

Für die Verbindung mehrerer Interruptquellen an den maskierbaren Interrupteingang INTR der CPU wird der programmierbare Interruptcontroller 8259A (PIC) eingesetzt, der speziell für das System 8086 entwickelt wurde (A-Typ erforderlich!).

Ein einzelner PIC kann 8 Interruptquellen verwalten und für diese die Prioritätsentscheidung übernehmen. Durch die Anschaltung von bis zu 8 Slave-PIC-Bausteinen an einen Master-PIC können maximal 64 unterschiedliche Interruptquellen im System verarbeitet werden.

Für eine über einen oder mehrere PICs ausgewählte Interruptanforderung wird der CPU-Eingang INTR auf '1' geschaltet. Die CPU reagiert nach dem Abschluß des in der Abarbeitung befindlichen Befehls mit zwei aufeinander folgenden Interruptbestätigungszyklen, die je einen INTA-Impuls an den PIC schalten (Bild 4.2).

Während des zweiten INTA-Impulses setzt der ausgewählte PIC einen der Interruptquelle zugeordneten Interruptvektor auf den niederwertigen Teil DBØ ... DB7 des Daten-BUS. Die CPU multipliziert diesen Vektor mit dem Wert 4. Damit entsteht ein Pointer auf die Adresse der Interrupt-Service-Routine in der Interrupttabelle. Die CPU übernimmt die Werte für das CS- und das IP-Register aus der Interrupttabelle, nachdem die Fortsetzungsadresse des unterbrochenen Programms mit Segment- und Offsetanteil im Stack abgespeichert worden ist. Zusätzlich wird im Stack das Flagregister, einschließlich des Interruptflags, gekellert. Das Interruptflag ist am Anfang der Interrupt-Service-Routine automatisch auf 'Ø' gesetzt worden.

Am Ende der Interrupt-Service-Routine werden die im Stack abgespeicherten Informa-

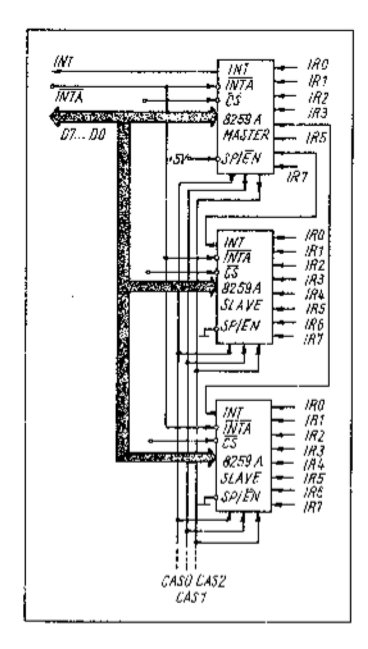


Bild 4.4 Kaskadierung von drei PICs

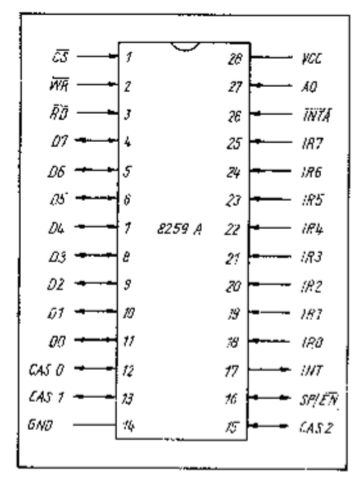


Bild 4.5 Pin-Belegung des 8259A

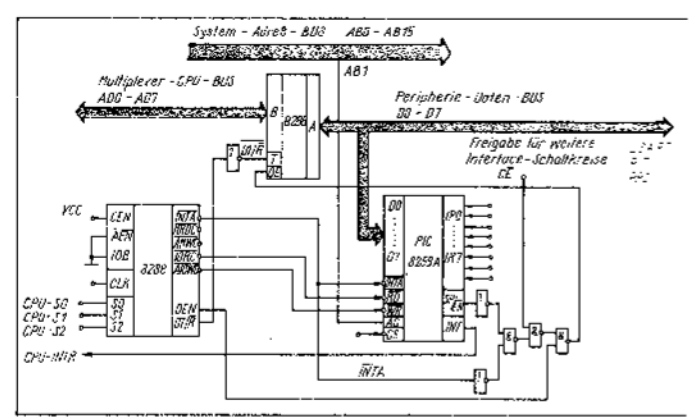


Bild 4.6 Anschaltung eines PIC in einen 8086-Mikrorechner (Maximum-Mode)

tionen mit dem Befehl IRET in das Flagregister, das IP- und CS-Register zurückgeschrieben. Damit ist der Zustand vor der Interruptannahme wieder hergestellt und mit dem Wert '1' für das Interruptflag eine erneute Interruptfreigabe der CPU gegeben.
Die Interrupt-Service-Routinen können auch mit Interruptbefehlen aufgerufen werden („Software-Interrupts"). Im 2-Byte-Interrupt-Befehl INT zz wird der zugehörige Interruptcode zz angegeben. Die Stackeintragung entspricht einem Hardware-Interrupt.

4.2. Aufbau und Funktionsprinzip des programmierbaren Interruptcontrollers 8259A

Nach Bild 4.3 enthält der PIC die Funktionseinheit für den System-BUS-Anschluß mit Daten-BUS-Puffer für den niederwertigen Teil DBØ ... DB7 des Daten-BUS und der Anschaltung der Steuersignale RD und WR. Das Signal CS selektiert den Baustein innerhalb des E/A-Adreßraumes. Mit dem Eingang AØ, der mit einer unteren Bitleitung des Adreß-BUS verbunden wird (z. B. AB1), werden die zwei möglichen Adressen für die Steuerkanäle des PIC unterschieden.
Die Prioritätsentscheidung für die 8 Eingangssignale an den Interrupt-Anforderungseingängen IRØ ... IR7 übernimmt eine Prioritätslogik im Zusammenwirken mit drei Registern.
Das Interruptrequestregister (IRR) speichert alle Interruptanforderungen an den Interruptrequesteingängen IRØ ... IR7. Die Interruptanforderung wird entweder mit der 'Ø/1'-Flanke (edge triggered mode) oder mit dem '1'-Pegel (level triggered mode) übernommen. Die Ausgänge des IRR werden mit dem Interrupt-Masken-Register (IMR) maskiert.
Innerhalb der Interruptbestätigung durch die CPU wird die höchstpriorisierte Anforderung bei Berücksichtigung der Maskierungsbedingung aus dem IRR mit dem ersten INTA-Impuls in ein In-Service-Register (ISR) übernommen, falls in diesem keine höherpriorisierte Interruptabarbeitung markiert ist. Im ISR-Register sind also alle in der Abarbeitung befindlichen Interrupts notiert. Das höchste gesetzte Bit des ISR wird entweder am Ende der Interrupt-Service-Routine mit einem speziellen Kommando an den PIC oder automatisch nach dem zweiten INTA-Impuls (AEOI-Mode des PIC) gelöscht.
Die Kaskadierungslogik gestattet die Anschaltung von Slave-PICs an einen Master-PIC. Nach Bild 4.4 werden alle PICs an den Daten-BUS, die Steuersignale RD und WR und die Adreßauswahlsignale angeschlossen. Die Interruptausgänge INT der Slave-PICs sind mit den IR-Eingängen des Master-PIC verbunden.
Die Betriebsart jedes PIC als Master oder Slave wird durch die Programmierung und den Anschluß SP eingestellt. Der in einem Slave-PIC ausgewählte höchstpriorisierte Interrupt wird an den IR-Eingang des Master-PIC gelegt und von diesem nach den Prioritätsbedingungen der Masterebene an den INTR-Eingang der CPU geschaltet.
Die Interruptbestätigungszyklen werden von allen PICs parallel ausgewertet. Der Master-PIC legt auf die Ausgänge CASØ, CAS1 und CAS2 den Identifikationscode desjenigen Slave-PICs der den Interruptvektor auf den Daten-BUS zu geben hat.

4.3. Anschlußbeschreibung des 8259A

Die Pin-Belegung des 8259A zeigt Bild 4.5. Die Anschlüsse haben die folgende ***Bedeutung:***

CS	Bausteinauswahl (chip select), Eingang
WR	Schreibsignal (write), Eingang
RD	Lesesignal (read), Eingang
D7–DØ	Datenleitungen für das Schreiben von Steuerkommandos und das Lesen von Statusinformationen und des Interruptvektors, bidirektional, tri-state
CASØ, CAS1, CAS2	Kaskadierungssignale, Ausgänge für Master-PIC, Eingänge für Slave-PICs
SP/EN	Programmierung im Puffermode: EN, Ausgang Ansteuersignal für Daten-BUS-Lesetreiber Programmierung im Nichtpuffermode: SP, Master/Slave-Selektion. Eingang SP = 1: Master SP = Ø: Slave
INT	Interruptausgang
IRØ–IR7	Interruptanforderungseingänge, pegel- oder flankengesteuert (interrupt request)
INTA	Interruptbestätigungssignal des Buscontrollers 8288 (interrupt acknowledge), Eingang
AØ	Portauswahlsignal, Eingang, Verbindung mit AB1 des Adreß-BUS üblich)
Vcc	+5 V
GND	Masse

Die schaltungstechnische Einordnung des PIC 8259A im System 8086 zeigt Bild 4.6.

4.4. Programmierung des 8259A

Der 8259A ist sowohl für den Einsatz in 8-Bit-Prozessorsystemen (8080, 8085) als auch für die 16-Bit-Prozessoren 8086 und 8088 geeignet. Im folgenden wird die Programmierung nur für die 16-Bit-Prozessoren erläutert.
Die Programmierung des PIC erfordert zuerst eine Grundinitialisierung mit 2 bis 4 **Initialisierungskommandos**. Ein erstes Initialisierungskommandowort *ICW1*, ausgegeben auf Steuerport mit AØ = Ø, startet die Programmierung. ICW1 enthält die Einstellung der Interruptrequesteingänge auf die Flanken- oder Pegelsteuerung und die Angabe, ob ein PIC oder mehrere kaskadierte PICs im System enthalten sind.
Das ebenfalls in allen Fällen notwendige zweite Initialisierungskommando *ICW2* enthält die 5 werthöchsten Bitstellen des Interruptvektors, der im zweiten Interruptbestätigungszyklus vom PIC auf den Daten-BUS gelegt wird. Der PIC setzt entsprechend der bestätigten Interruptanforderung die Bitpositionen TØ, T1 und T2 des Interruptvektors ein.

wird fortgesetzt

Literatur:
/1/ Peripheral Design Handbook, Intel 1978

Mikroprozessorsystem K 1810 WM86

Hardware · Software · Applikation (Teil 3)

Prof. Dr. Bernd-Georg Münzer
(wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann,
Wolfgang Kabatzke, Frank Kamrad,
Helfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock,
Sektion Technische Elektronik,
Wissenschaftsbereich Mikrorechentechnik/
Schaltungstechnik

Tafel 4.1 Initialisierungskommandos

ICW 1 *Portadresse mit A0 – 0*

D7	D6	D5	D4	D3	D2	D1	D0
[illegible]	[illegible]	[illegible]	1	LTIM	[illegible]	SNGL	IC4

LTIM *Eingangssteuerung für IR-Eingänge*

1 *pegelgesteuert ('1'-aktiv)* (level triggered mode)
0 *flankengesteuert ('01'-Flanke)* (edge triggered mode)

SNGL *Einzel- oder Multi-PIC-System*

0 *kaskadierte PICs, ICW3 erforderlich*
1 *Einzel-PIC, ICW3 nicht erforderlich*

IC4 *Ausgabe von ICW4*

0 *nicht erforderlich*
1 *erforderlich*

ICW2 *Portadresse mit A0 = 1*

D7	D6	D5	D4	D3	D2	D1	D0
T7	T6	T5	T4	T3	[illegible]	[illegible]	[illegible]

T7–T3 *höherwertiger 5-Bitanteil des Interruptcode*

ICW3 *Portadresse mit A0 = 1*
Mastermode:

D7	D6	D5	D4	D3	D2	D1	D0
S7	S6	S5	S4	S3	S2	S1	S0

S7–S0 *Slave-PIC-Anschluß*

0 *einfache Interruptquelle*
1 *an IR-Eingang ist ein Slave-PIC angeschlossen*

Slavemode:

D7	D6	D5	D4	D3	D2	D1	D0
x	x	x	x	x	ID2	ID1	ID0

ID2–ID0 *Identifikationscode für Slave-PIC*

ICW4 *Portadresse mit A0 = 1*

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	SFNM	BUF	M/S	AEOI	1

AEOI *Interruptfreigabeform*

0 *Rücksetzen des ISR-Bits mit EOI-Kommando (Normal EOI)*
1 *automatisches Rücksetzen der höchstpriorisierten ISR-Bitstelle nach Interruptannahme (nur im Mastermode)*

BUF *Puffer-Mode*

0 *keine Puffersteuerung*
1 *Steuerausgang für Daten-BUS-Treiber auf SP / EN*

M/S *PIC-Arbeitsweise im Puffer-Mode (BUF = 1)*

0 *Slave-Mode*
1 *Master-Mode*

SFNM *Interruptverschachtelungsmode*

0 *keine Interruptverschachtelung für Anforderungen innerhalb eines Slave-PIC*
1 *volle Interruptverschachtelung bei kaskadierten PICs, (special full nested mode)*

Tafel 4.1 enthält die Initialisierungskommandos **ICW1** bis **ICW4**.
Im **AEOI**-Mode (automatic end of interrupt) wird die der laufenden Interrupt-Service-Routine zugeordnete Bitstelle im ISR bereits mit der fallenden Flanke des zweiten **INTA**-Impulses zurückgesetzt. Dadurch kann auf das Rücksetzen am Ende des Interruptprogramms mit einem speziellen **PIC**-Kommando verzichtet werden. Innerhalb einer prioritätsgesteuerten verschachtelten Interruptstruktur ist dieser Mode jedoch nicht sinnvoll.
Nach der Einstellung des Puffermodes erzeugt der **PIC** am Ausgang **SP/EN** für die Dauer der Ausgabe des Interruptvektors ein Steuersignal zur Ansteuerung der Daten-**BUS**-Puffer. Im Puffermode legt die Bitstelle **M/S** die Reaktion als Master- oder Slave-**PIC** fest.
Mit **Operationssteuerkommandos** (operation control word, OCW) lassen sich weitere Betriebsarten des PIC jederzeit während der Programmabarbeitung einstellen (Tafel 4.2). Operationssteuerworte können in beliebiger Reihenfolge ausgegeben werden.

Tafel 4.2 Operationskommandos

OCW1 *Portadresse mit A0 = 1*

D7	D6	D5	D4	D3	D2	D1	D0
M7	M6	M5	M4	M3	M2	M1	M0

M7–M0 *Interrupteingangsmaskierung*

0 *Interrupteingang freigegeben*
1 *Interrupteingang gesperrt*

OCW2 *Portadresse A0 – 0*

D7	D6	D5	D4	D3	D2	D1	D0
R	SL	EOI	0	0	L2	L1	L0

L7–L0 *Binärcode für Interrupteingangsnummer*

R SL EOI *Kommando*

R SL EOI	
0 0 1	*EOI-Kommando (end of interrupt) mit L0, L1, L2 – 0*
0 1 1	*SEOI-Kommando (special end of interrupt)*
1 0 1	*Rotation im EOI-Mode mit L0, L1, L2 = 0*
1 0 0	*Rotation im AEOI-Mode, Setzen mit L0, L1, L2, – 0*
0 0 0	*Rotation im AEOI-Mode, Löschen mit L0, L1, L2 = 0*
1 1 1	*Rotation im SEOI-Mode*
1 1 0	*Priorität setzen*
0 1 0	*keine Operation*

OCW3 *Portadresse mit A0 = 0*

D7	D6	D5	D4	D3	D2	D1	D0
x	ESMM	SMM	0	1	P	RR	RIS

RR RIS *Statuslesekommando mit D2, D5, D6 – 0*

RR RIS	
1 0	*nächster Lesebefehl bezieht sich auf IRR*
1 1	*nächster Lesebefehl bezieht sich auf ISR*
0 x	*keine Leseoperation*

P *Pollingkommando mit D0, D1, D5, D6 – 0*

1 *eingeschaltet*
0 *ausgeschaltet*

ESMM SMM *spezieller Maskierungsmode mit D0, D1, D5, D6 = 0*

ESMM SMM	
1 1	*speziellen Maskierungsmode einstellen*
1 0	*speziellen Maskierungsmode abstellen*
0 x	*keine Maskierungsmodeeinstellung*

Das Operationssteuerwort **OCW1** setzt das Interruptmaskenregister. Auch für gesperrte Interrupteingänge werden jeoch bei flankengesteuerten Eingängen Interruptanforderungen im IRR abgespeichert.
Mit dem Operationssteuerwort **OCW2** werden sieben Kommandos der Interruptfreigabe und der Interruptorganisation gebildet: Die Ausgabe eines **EOI**-*Kommandos* (end of interrupt) setzt die höchstpriorisierte aktive Bitstelle des **ISR** zurück.
Das **SEOI**-*Kommando* (specific end of interrupt) bezieht sich auf eine ausgewählte Bitstelle des **ISR**.
Nach der Initialisierung sind die Prioritäten den **IR**-Eingängen fest zugeordnet, wobei der Eingang **IR0** die höchste Priorität besitzt. Diese Prioritätsorganisation kann durch die Rotationskommandos abgeändert werden. Durch die Rotation der Prioritäten erhalten alle **IR**-Eingänge die gleichen Zugriffsmöglichkeiten. Die Interruptrotation ist den verschiedenen Formen der Interruptfreigabe des **IRR** (**EOI**-, **SEOI**- und **AEOI**-Mode) in unterschiedlichen Kommandos zugeordnet.
Das Kommando *Rotation im* **EOI**-*Mode* veranlaßt nach dem Rücksetzen der höchstpriorisierten Bitstelle des **ISR** die Zuordnung der niedrigsten Priorität (Wert 7) für den zugehörigen **IR**-Eingang. Die diesem Eingang folgenden Eingänge erhalten, beginnend mit Priorität 0, abnehmende Prioritäten zugewiesen. Auf diese Weise wird gesichert, daß eine erneute Interruptanforderung für einen **IR**-Eingang erst wieder berücksichtigt wird, wenn alle anderen anfordernden Eingänge bedient worden sind. Die höchste Interruptpriorität rotiert nach jeder Interruptbearbeitung zum nächsthöheren **IR**-Eingang.
Die Prioritätsrotation erfolgt im **AEOI**-Mode automatisch nach dem zweiten **INTA**-Impuls bei der Interruptannahme. Mit den Kommandos *Rotation im* **AEOI**-*Mode; Setzen* und *Rotation im* **AEOI**-*Mode; Löschen* wird die Rotation im AEOI-Mode ein- und ausgeschaltet.
Die niedrigste Priorität kann einem IR-Eingang mit dem Kommando *Priorität setzen* zugewiesen werden. Damit wird zugleich für alle folgenden Eingänge, beginnend mit der höchsten Priorität, eine abnehmende Priorität eingestellt. Diese Neueinstellung wird erst für die danach eintreffenden Interruptanforderungen wirksam.
Das Kommando *Rotation im* **SEOI**-*Mode* kombiniert die Funktionen der Kommandos für das Einstellen des **SEOI**-Modes und des Interruptsetzens. Die ausgewählte Bitstelle des **IRR** wird gelöscht und dem zugehörigen Eingang die niedrigste Priorität zugewiesen. Alle folgenden **IR**-Eingänge erhalten neue Prioritäten in der o. a. Form.

Leseoperationen:
Die Register **IRR**, **ISR** und **IMR** können gelesen werden. Vor einer Leseoperation für die Portadresse mit A0 = 0 muß der Zugriff auf **IRR** oder **ISR** durch eine Einstellung mit dem Kommandosteuerwort **OCW3** ausgewählt

werden. Diese Auswahl bleibt bis zur Neueinstellung auch über mehrere Leseoperationen erhalten. Nach der Initialisierung wird der Zugriff auf **IRR** eingestellt.
Um das Maskenregister IMR zu lesen, ist eine Voreinstellung nicht notwendig, da dafür die Portadresse mit **A0** = 1 benutzt wird.
Durch **OCW3** kann auch ein *spezieller Maskierungsmode* eingestellt werden. In diesem Mode werden Interruptbearbeitungen für Anforderungen mit einer Priorität, die geringer ist als die der in der Abarbeitung befindlichen Interruptoutine, ermöglicht. Weitere Anforderungen des bearbeiteten **IR**-Eingangs werden jedoch nicht bedient, um eine unkontrollierte Verschachtelung desselben Interruptprogramms zu vermeiden.

Tafel 4.3 Programmierbeispiel

```
CODSEG          EQU  1000H
INTOFF          EQU  0E8H            ;Adresse in der
                                     ;Interrupttabelle
;
PIC0            EQU  0C0H            ;PIC-Adresse mit A0=0
PIC1            EQU  0C2H            ;PIC-Adresse mit A0=1
PICICW1         EQU  17H             ;Einzel-PIC,
                                     ;flankengesteuert
PICICW2         EQU  38H             ;Interruptvektor
PICICW4         EQU  0FH             ;Master- und Buffer-
                                     ;mode. AEOI
PICOCW1         EQU  0FBH            ;Interruptfreigabe
                                     ;für IR2
;
TIMCH0          EQU  0D0H            ;Timeradresse für
                                     ;Kanal 0
TIMCON          EQU  0D6H            ;Timeradresse für
                                     ;Steuerkanal
TIMMD0          EQU  36H             ;Steuerwort Timer
                                     ;(Mode 3, dual)
TIMOL           EQU  02H             ;Zeitkonstante,
                                     ;unterer Anteil
TIMOH           EQU  30H             ;Zeitkonstante,
                                     ;oberer Anteil
USART_CONTROL      EQU   0DAH        ;Portadresse USART,
                                     ;Control
USART_DATA EQU  0D8H                 ;Portadresse USART,
                                     ;Daten
                DSEG 0               ;Adreßeinstellung in
                ORG  INTOFF          ;Interruptadreßtabelle
                DW   OFFSET ISR      ;Adresse der Inter-
                DW   CODSEG          ;rupt-Service-Routine

                CSEG CODSEG
TEST:           CLI                  ;CPU-Interrupt sperren
                MOV  AL,PICICW1      ;PIC-Initialisierungs-
                                     ;kommandos
                OUT  PIC0,AL         ;ausgeben
                MOV  AL,PICICW2
                OUT  PIC1,AL
                MOV  AL,PICICW4
                OUT  PIC1,AL
                MOV  AL,PICOCW1      ;PIC-Interruptmaske
                                     ;ausgeben
                OUT  PIC1,AL
                MOV  AL,TIMMD0
                OUT  TIMCON,AL       ;Mode-Einstellung
                                     ;Timer
                MOV  AL,TIMOL        ;Ausgabe der Zeitkon-
                                     ;stante
                OUT  TIMCH0,AL       ;für Timer
                MOV  AL,TIMOH
                OUT  TIMCH0,AL
                STI                  ;CPU-Interrupt frei-
                                     ;geben
LP:             JMPS LP              ;endlose Schleife
;
ISR:            MOV  AL,'*'
                CALL OUTPUT          ;Bildschirmausgabe
                IRET
;
OUTPUT:         PUSH AX
OUTPUT1:        IN   AL,USART_CONTROL
                                     ;Lesen Statusregister
                TEST AL,1            ;Prüfen TxRDY = 1
                JZ   OUTPUT1
                POP  AX              ;Sendepuffer leer!
                OUT  USART_DATA,AL   ;Daten schreiben
                RET
```

Der **PIC** kann auch für die Unterstützung von Pollingverfahren eingesetzt werden. Nach der Ausgabe des *Polling-Kommandos* kann mit einem einzigen Lesebefehl festgestellt werden, ob an einem **IR**-Eingang des PIC eine Bedienanforderung anliegt. Dieser Fall wird mit **D7** = 1 des eingelesenen Bytes gekennzeichnet. Die drei unteren Bits dieses Bytes enthalten die IR-Eingangsnummer der höchstpriorisierten Anforderung.

4.5 Programmbeispiel

Das Programmbeispiel in Tafel 4.3 behandelt die Anschaltung eines Programmierbaren Intervall-Timers **8253** an den **PIC.** Dabei wird der Ausgang des Zählers 0 mit dem Eingang **IR2** des **PIC** verbunden.
Der Timer erzeugt im Mode 3 eine Impulsfolge im Zeitabstand von 10 ms (s. Abschn. 3).
Die **PIC**-Progammierung enthält innerhalb eines Einzel-**PIC**-Systems die Interruptfreigabe für den Eingang **IR2** mit Flankensteuerung. Aus der Angabe des Interruptvektors **38H** ergibt sich für den Eingang **IR2** die Adresse **0:0E8H** innerhalb der Interruptadreßtabelle. Auf dieser Speicherposition wird die Adresse des Interrtupt-Service-Programms angegeben.
Nach der Programmierung des Timers und des **PIC** verbleibt das Hauptprogramm in einer endlosen Schleife. Im Abstand von 10 ms wird die Interrupt-Service-Routine eingeblendet, in der das Zeichen '*' auf dem Bildschirm ausgegeben wird. Die Programmformulierung enthält die mnemonischen Befehlsbeschreibungen und Pseudoanweisungen des Assembler-Programmes **ASM86**, die im Abschnitt 5 dieser Folge beschrieben werden.

5. Assemblerbefehle der 8086-CPU

5.1 Befehlsübersicht

Der Befehlssatz der **8086-CPU** enthält die wichtigsten, aus der 8-Bit-Technik bekannten Funktionsgruppen, welche jedoch durch eine Reihe weiterer effektiver Befehle ergänzt wurden.
Die CPU realisiert die Verarbeitung von 8-Bit-Daten (Bytes) und 16-Bit-Daten (Wörter).
Arithmetische Operationen beinhalten verschiedene Formen der Addition und Subtraktion, der Multiplikation und der Division.
Logische Operationen beziehen sich auf **UND-**, **ODER-** und **XOR**-Verknüpfungen zwischen zwei Operanden und die Negation, Rotation und Verschiebung von Einzeloperanden.
Stringoperationen gestatten das Umladen und den Vergleich von Speicherblöcken und die Abspeicherung bzw. Suche von Zeichen in Speicherbereichen.
Für die Programmablauforganisation steht eine Vielzahl unbedingter und bedingter Sprünge, Schleifenanweisungen, direkte und indirekte Unterprogrammaufrufe und Returnanweisungen zur Verfügung.
Die Registerstruktur des 8086 zeigen die Bilder 2.3, 2.4 und 2.5.
Die vier Hauptregister **AX**, **BX**, **CX** und **DX** können als 16-Bit-Register oder 8-Bit-Register mit den Bezeichnungen **AL**, **AH**, **BL**, **BH**, **CL**, **CH**, **DL** und **DH** benutzt werden. Die Indexregister **SI** (source index register) und **DI** (destination index register) und das Pointerregister **BP** (base pointer) werden als 16-Bit-Operandenregister oder für die indirekte Speicher-Adressierung vewendet. Der Stackpointer **SP** verwaltet den Prozessorstack.
Die Hauptregister, Indexregister und Pointerregister können in der Mehrzahl aller Befehle als Operandenregister verwendet werden.

5.2 Assemblerprogrammierung

Im folgenden werden alle Befehle auf dem Niveau der Assemblerbeschreibung erläutert, wobei auf das Assemblerprogramm **ASM86** im Betriebssystem **SCP 1700** Bezug genommen wird, das mit dem gleichnamigen Cross-Assembler im Betriebssystem **SCPX** übereinstimmt.
Auf Besonderheiten des Assemblerprogrammes **RASM86** im Betriebssystem **SCP 1700** wird im Abschnitt 6 hingewiesen.
Die Beschreibung der Befehle erfolgt in mnemonischer Darstellung, wobei die Operanden in Registern mit dem Registernamen dargestellt sind.
Konstanten können in dualen, oktalen, dezimalen und hexadezimalen Darstellungen oder im **ASCII**-Code auftreten.

5.2.1 Speicheradressierung

Bei der *direkten* Adressierung von Speicherplätzen werden konstante Adressen numerisch oder als Symbol angegeben (Bild 5.1.a).
OFFSET = 16 Bit offset (im Befehl).
Bei der *indirekten* Adressierung ist
• die Offsetadresse in einem der 16-Bit-Register enthalten:
– Basisregister **[BX]**, **[BP]** oder
– Indexregister **[SI]**, **[DI]**.
Die Registerangabe erfolgt in eckigen Klammern (Bild 5.1.b).
OFFSET = **[BX]** oder **[BP]** oder **[SI]** oder **[DI]**.
• die Offsetadresse als Summe von zwei 16-Bit-Registern angegeben:
– Basisregister + Indexregister
OFFSET = **[BX+SI]** oder **[BX+DI]** oder **[BP+SI]** oder **[BP+DI]** (Bild 5.1.c)

Bild 5.1 Adressierungsarten des 8086
a) Direktoperand
b) Operand in Indexregister SI,DI oder Basisregister BX,BP
c) Operand in Summe von Indexregister plus Basisregister

• eine zusätzliche positive oder negative Verschiebung (displacement) wird im Assemblerprogramm **ASM86** außerhalb der eckigen Klammern ergänzt, während einige Reassembler, z. B. im Monitor des **A 7100**, dafür die sinnvollere Darstellung aller Bestandteile der Adresse innerhalb der eckigen Klammern benutzen.

Beispiel:

OFFSET = [**SI** + **DI**] + displacement

Die *Standard-Zuweisung* der Operanden zu den Segmentregistern lautet:

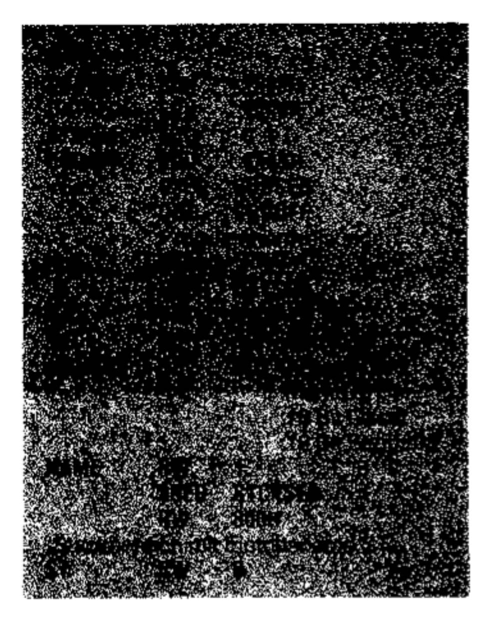
Operand	*Segmentregister*
Direktoperand, **BX**, **SI**, **DI**	Datensegment **DS**
SP, **BP**	Stacksegment **SS**
DI (Stringoperationen)	Extrasegment **ES**

Wenn ein Indexregister zusammen mit einem Basisregister benutzt wird, gilt das Standard-Segmentregister des Basisregisters als Standardsegment für den gesamten Operanden:
[**BP** + **SI**] mit Segment **SS**
[**BX** + **DI**] mit Segment **DS**

Speicheradreßangaben mit vom Standard abweichenden Segment-Registern werden durch Vorsetzen des neuen Segmentregisternamens dargestellt (Segmentpräfix).
Der ASM86 verarbeitet über den Umfang der Prozessorbefehle hinaus sogenannte *Pseudoanweisungen*, wovon nur einige wichtige hier angegeben sind.
Mit der **EQU**-Anweisung werden Symbole (Namen) für numerische Konstanten eingeführt. Diese können auch über einfache Ausdrücke definiert werden.



Mit den Pseudoanweisungen **DB** (define byte), **DW** (define word), **RB** (reserve byte) und **RW** (reserve word) können im Codesegment, Datensegment, Stacksegment und Extrasegment Speicherbereiche definiert und benannt werden. Die Anweisungen DB und DW initialisieren den Speicherbereich mit den in den anschließenden Listen byte- oder wortweise angegebenen Konstanten. Hinter RB und RW steht nur die Zahl der reservierten Bytes oder Wörter.
Die **DD**-Anweisung initialisiert vier Bytes im Speicher mit einer vollständigen Offset- und Segmentadresse.
Die Segmentzuordnung zur definierten Adresse für Programm- und Datenbereiche erfolgt im Assemblerprogramm ASM86 mit den Pseudoanweisungen **CSEG**, **DSEG** **SSEG** und **ESEG** für das Code-, Daten-, Stack- und Extrasegment.
Der Inhalt des **CS**-Registers wird beim Programmstart eingestellt, während die restlichen Segmentregister *vor* ihrer Verwendung im Programm geladen werden müssen.
Mit der Anweisung **ORG** wird die Offsetadresse innerhalb des Segments festgelegt.



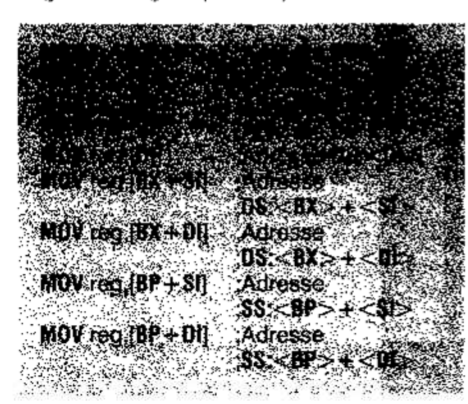
5.3 8086-Befehle

5.3.1 Transportbefehle

MOV-Befehle dienen dem Transport von 8- und 16-Bit-Daten zwischen den Registern untereinander, zwischen Registern und Speicherplätzen und von 8- und 16-Bit-Konstanten in Register und auf Speicherplätze.
Als 16-Bit-Register in **MOV**-Befehlen sind die vier Hauptregister, die Indexregister, die Pointerregister und mit Einschränkungen die Segmentregister zugelassen.
Die folgenden Beispiele vermitteln die Registerschreibweise in Register-Register-Befehlen in Assemblernotierung:



Bei MOV-Befehlen sind für den Speicherzugriff die im Abschnitt 5.2.1 genannten direkten und indirekten Adressierungsarten mit den Standard-Zuweisungen zu den Segmentregistern möglich (Bild 5.1):

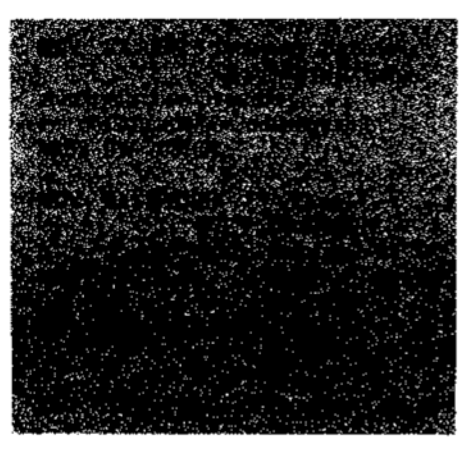


```
MOV reg,[BX+DI]   ;Adresse
                  DS:<BX>+<DI>
MOV reg,[BP+SI]   ;Adresse
                  SS:<BP>+<SI>
MOV reg,[BP+DI]   ;Adresse
                  SS:<BP>+<DI>
```

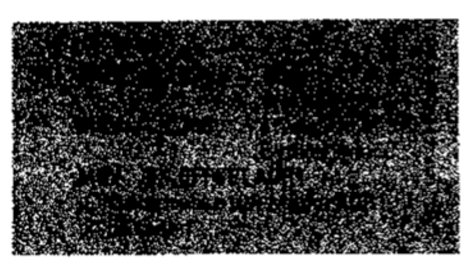
Die folgenden *Beispiele* für den Transfer zwischen Registern und Speicherinhalten geben die Assemblernotierung für die verschiedenen Adressierungsformen an. Aus der Art des beteiligten Registers ergibt sich ein Byte- oder Wortzugriff auf den Speicher.



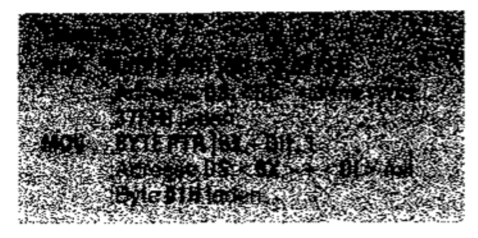
Bei Angabe von Adreß-Symbolen, die nicht im Datensegment definiert sind, erzeugt der Assembler automatisch ein entsprechendes Segment-Override-Präfix.



Für die Definition von im Befehl angegebenen Konstanten ist die Breite des Zielregisters entscheidend.



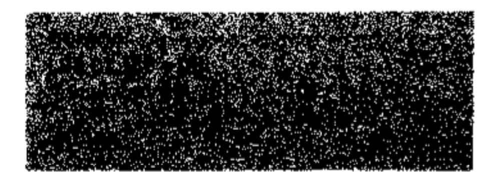
MOV-Befehle für das Einschreiben von *Konstanten* in den Speicher oder Befehle mit nur einem Operanden, für den auf den Speicher zugegriffen werden muß, benötigen in der Assemblernotierung die Angabe zu einem Byte- oder einem Worttransfer. Das erfolgt durch den Zusatz **BYTE PTR** oder **WORD PTR** vor der Angabe der Speicheradresse
Diese Angaben sind auch bei einer Reihe anderer Befehle mit Konstanten-Darstellung erforderlich.



Mit Adreßtransfer-Operationen können Offsetwerte und auch Segmentwerte in Register geladen werden.
Der Befehl **LEA** (load effextive address) schreibt die Offsetadresse in eines der acht möglichen 16-Bit-Register.

Beispiel:
LEA BX,[BX + SI + 0A67H]

Die Befehle **LDS** und **LES** übertragen den Inhalt von 2 benachbarten Speicherplätzen in ein 16-Bit-Register und den Inhalt der beiden nachfolgenden Speicherplätze in **DS** bzw. **ES**.



Der Spezialfall eines Transportbefehls **XLAT** (translate) dient zur Übertragung des Inhaltes eines Tabellenelements. Dabei wird aus der Summe der Inhalte von BX (Startadresse der Tabelle) und **Al** (displacement) eine Offsetadresse gebildet, mit der ein Byte vom Datensegment in **AL** geladen wird.



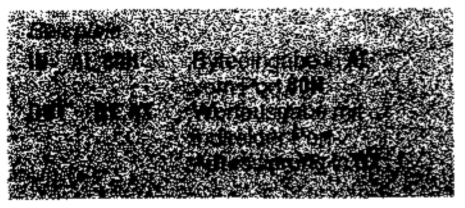
XCHG-Befehle (exchange) realisieren den Austausch von Bytes oder Wörtern zwischen Registern oder zwischen einem Rtegister und einem Speicherpiatz.

Beispiel:
XCHG SI,[DI]

PUSH-und **POP**-Befehle transferieren Wortinformationen zwischen einem Register und dem Prozessorstack. Vor jedem Speicherschreiben wird die Adresse im **SP**-Register dekrementiert und nach jedem Lesen inkrementiert. Als zu transferierende 16-Bit-Register sind neben den Hauptregistern, den Indexregistern und Pointerregistern auch die Segmentregister zugelassen.
Die speziellen Befehle **PUSHF** und **POPF** beziehen sich auf die Stackoperationen für das Flagregister.

Beispiele:
PUSH DS ;DS in Stack
POP BX ;BX aus Stack
PUSHF
POPF

Die Flagladeoperation **LAHF** (load AH with flags) überträgt die dem Prozessor **8080** entsprechenden Flags des **8086** in der Anordnung **SF,ZF,x,AF,x,PF,x,CF** in das Register **AH**.
Der Befehl **SAHF** (store AH into flags) realisiert die entsprechende Transportoperation für die Gegenrichtung. Die **IN**- und **OUT**-Befehle sind 8- und 16-Bit-Transportbefehle zwischen **AL** oder **AX** und Ein-/Ausgabekanälen. Dabei kann die Kanaladresse als Bytekonstante im Befehl oder als Wort-Variable im Register **DX** angegeben werden.



Mit den folgenden Befehlen werden einzelne Binärstellen des Flagregisters eingestellt:

CLC (clear **CF**)	;<**CF**>=**0**
CMC (complement **CF**)	;<**CF**>=<$\overline{\textbf{CF}}$>
STC (set **CF**)	;<**CF**>=**1**
CLD (clear **DF**)	;<**DF**>=**0**
STD (set **DF**)	;<**DF**>=**1**
CLI (clear **IF**)	;<**IF**>=**0**, Sperren ;maskierbarer ;Interrupts
STI (set IF)	;<**IF**>=**1**, Freigabe ;maskierbarer ;Interrupts

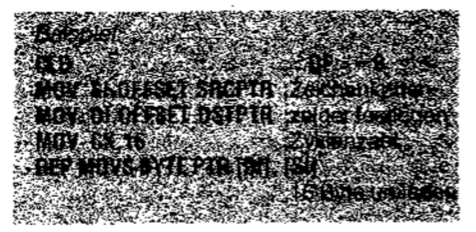
Eine andere Möglichkeit zur Einstellung aller Flags besteht in der Abspeicherung des gewünschten Bitmusters im Stack und dessen Übernahme in das Flagregister mit dem Befehl **POPF**.

5.3.2 Zeichenkettenbefehle

Zeichenkettenbefehle dienen dem Umladen und Vergleich von bis zu 64 KByte großen Speicherbereichen bzw. der Zeichensuche und Zeicheneintragung in Speicherbereichen. Für die entsprechenden String-*Einzeloperationen*

MOVS	move string
CMPS	compare string
STOS	store string
LODS	load string
SCAS	scan string

kann durch die Angabe des Wiederholungspräfix **REP** eine zyklische Abarbeitung festgelegt werden, für die das Register **CX** die Zyklenzahl enthält. Mit dem Präfix **REP** wird bei den Befehlen **CMPS** und **SCAS** eine zweite Wiederholbedingung des zyklischen Ablaufes in Abhängigkeit vom Inhalt des Z-Flags durchgeführt. Es erfolgt dabei eine automatische Wiederholung unter der Bedingung **CX** ungleich Null und <**ZF**>=1.
Einheitlich für alle Zeichenkettenbefehle gilt, daß die *Quellzeichenkette* mit der Offsetadresse im Register *SI* und der Segmentadresse in *DS* und die *Zielzeichenkette* mit der Offsetadresse in *DI* und der Segmentadresse in *ES* definiert ist. Dadurch sind Zeichenkettenoperationen mit unterschiedlichen Speichersegmenten möglich.
Für eine eindeutige Übertragung ist die Angabe **BYTE PTR** bzw. **WORD PTR** erforderlich.
Der Befehl **MOVS** überträgt ein Byte oder Wort aus der Quellzeichenkette in die Zielzeichenkette. Bei Abarbeitung des Befehls werden die Zeiger *SI* und *DI* für beide Zeichenketten in Abhängigkiet vom Datenformat um *1* und 2 verändert. Ein Zeigerinkrement oder -dekrement erfolgt in Abhängigkeit vom Wert des **D**-Flags (**DF=1** Dekrement, **DF=0** Inkrement).
Der **MOVS**-Befehl beeinflußt keine Flags.

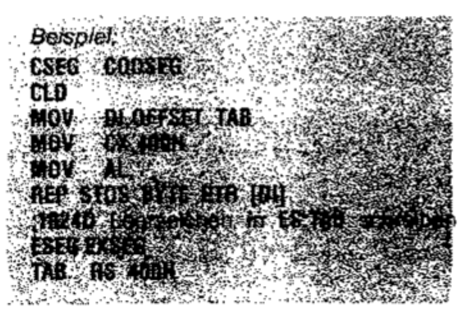


Im Befehl **CMPS** werden die Quell- und Zielzeichenkette byte- oder wortweise verglichen und die Zeiger wie beim **MOVS**-Befehl verändert. Das Ergebnis des Vergleichs steht in den Flags **AF**, **CF**, **OF**, **PF**, **SF** und **ZF**. Der von der Z-Flagbedingung abhängige zyklische Vergleich ergibt die Zeiger **DI**, **SI** auf die nächste Adresse der ersten Zeichenabweichung oder -übereinstimmung.

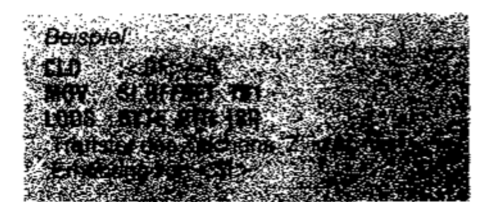
REP, **REPZ**,	Wiederholung bei <**ZF**>=1
REPNZ,	Wiederholung bei <**ZF**>=0



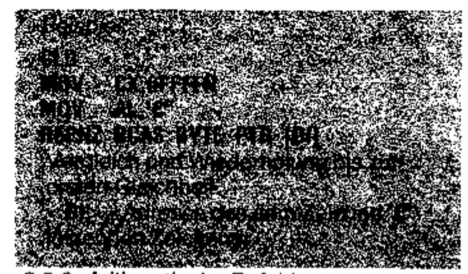
Mit dem Befehl **STOS** kann der Zielbereich im Extrasegment mit in **AL** oder **AX** vorgegebenen Bytes/Wörtern beschrieben werden.


Beispiel:
CSEG CODSEG
CLD

Die umgekehrte Operation führt der Befehl **LODS** aus. Ein Byte/Wort wird aus der Quellzeichenkette vom Datensegment in AL/AX übertragen.



Für die Suche eines in **AL/AX** vorgegebenen 8-/16-Bit-Zeichens in einer Zielzeichenkette im Extrasegment ist der Befehl **SCAS** in zyklischer Ausführung mit dem Präfix REPNZ geeignet.



5.3.3 *Arithmetische Befehle*

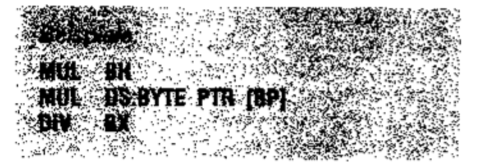
Die arithmetischen Operationen realisieren die vier Grundrechenarten für 8-/16-Bit-Dualdarstellungen und zweistellige Dezimalzahlen in unterschiedlichen Versionen. Operanden können in 8-/16-Bit-Registern oder im Speicher vorgegeben werden. Das Ergebnis kann in jedem dieser Register oder im Speicher abgelegt werden.
Im Ergebnis arithmetischer Operationen werden zusätzlich die folgenden Flageinstellungen realisiert:

- **CF = '1'**, wenn Überlauf bei Addition oder Unterlauf bei Subtraktion für das gesamte Format auftritt
- **ZF = '1'**, wenn Ergebniswert = 0
- **SF = '1'**, wenn höchstwertigstes Bit des Ergebnisses = 1
- **PF = '1'**, wenn Parität der unteren 8 Bit des Ergebnisses gerade
- **AF = '1'**, wenn bei Addition Überlauf oder bei Subtraktion Unterlauf zwischen den unteren Halbbytes des Ergebnisses auftritt
- **OF = '1'**, wenn ein Überlauf aus der zweithöchsten Stelle auf die werthöchste Stelle des Ergebnisses entsteht

Die Addition und Subtraktion von Dualdaten erfolgt in den Befehlen **ADD**, **SUB**, **ADC** (addition with carry) und **SBB** (subtraction with borrow) ohne oder mit Berücksichtigung des C-Flags. Negative Zahlen werden in der Zweierkomplementform dargestellt.
Sonderfälle der Addition und Subtraktion sind die **INC**- und **DEC**-Operationen mit der Erhöhung oder Erniedrigung des Inhaltes um 1 von 8-/16-Bit-Registern oder von Dualzahlen im Speicher.
Eine duale Subtraktion wird beim **CMP**-Befehl ausgeführt, wobei das Ergebnis nur die o. a. Flags beeinflußt.

Beispiele:
```
ADD  [SI],CX
CMP  CS:BYTE PTR [DI]+3,'A'
INC  WORD PTR [BX]
```

Der Befehl **NEG** erzeugt die Zweierkomplementdarstellung des Vorgabeoperanden.
Die dezimale Addition und Subtraktion werden über Korrekturoperationen im Anschluß an die dualen Operationen ausgeführt.
Der Befehl **DAA** (decimal adjust for addition) wandelt das duale Additionsergebnis im Register **AL** in eine gepackte Dezimaldarstellung um. Für die 3stellige dezimale Ergebnisdarstellung wird das C-Flag mitbenutzt.
Im Anschluß an die duale Addition einer zweistelligen ungepackten Dezimalzahl im ASCII-Code im Register AX mit einer einstelligen Dezimalzahl in Dualdarstellung erzeugt der Befehl **AAA** (**ASCII** adjust for addition) eine ASCII-Darstellung des Ergebnisses in **AX**.
Die entsprechenden Korrekturoperationen für die Subtraktion lauten **DAS** (decimal adjust for subtraction) und **AAS** (**ASCII** adjust for subtraction).
Eine bedeutende Verbesserung gegenüber der 8-Bit-Prozessor-Technik bietet beim **8086** die Hardwarerealisierung der Multiplikation und Division.
Der Multiplikationsbefehl **MUL** führt eine vorzeichenlose Multiplikation zwischen dem Register **AL** bzw. **AX** und einem Faktor in einem Register oder im Speicher aus. Im Fall einer 8-Bit-Multiplikation wird der über 8 Binärstellen reichende Anteil des Produktes in **AH** abgespeichert. Bei einer 16-Bit-Multiplikation kann das Ergebnis maximal 32 Binärstellen einnehmen, von denen die oberen 16 im Register **DX** abgelegt werden. Ist die Ergebnislänge größer als das Vorgabeformat, werden die Flags **CF=OF='1'** gesetzt. Alle anderen Flags sind nicht signifikant.
Der Befehl **IMUL** (integer multiply) verarbeitet vorzeichenbewertete Dualzahlen.
Für die Multiplikation von ungepackten Dezimalzahlen existiert die Korrekturoperation **AAM** (**ASCII** adjust for multiply) für die Nachbehandlung einer 8-Bit-Dualmultiplikation in **AX**.
Bei dem Befehl **DIV** muß ein ganzzahliger Dividend doppelter Länge durch einen Divisor einfacher Länge dividiert werden. Der Dividend wird bei der 8-Bit-Division im Register **AL** mit der Erweiterung in **AH** vorgegeben. Der Quotient steht nach der Operation in **AL** und der Rest in **AH**.
Bei der 16-Bit-Division wird der 32-Bit-Dividend in **AX** mit der Erweiterung in **DX** angegeben. Im Ergebnis stehen der Quotient in **AX** und der Divisionsrest in **DX**. Auf diese Weise können Divisionsbefehle verkettet werden, um Dualzahlen beliebiger Länge durch 8- oder 16-Bit-Zahlen zu dividieren.
Bei der Division wird im Fall eines Ergebnisüberlaufes (divide by zero) ein nicht maskierbarer Software-Interrupt auf eine in der Interrupttabelle stehenden Absolutadresse ausgelöst.
Der Befehl **IDIV** (integer division) führt die entsprechende Divisions-Operationen an vorzeichenbewerteten Operanden aus.



Nach einer Dualdivision von bis zu zweistelligen ungepackten Dezimalzahlen erzeugt der Befehl **AAD** (**ASCII** adjust for division) die **ASCII**-Darstellung des Quotienten.
Für Formatverlängerungen von vorzeichenbewerteten Dualzahlen in **AL** ergänzt der Befehl **CBW** (convert byte to word) das Erweiterungsregister **AH** bei positiven Werten mit **00H**, andernfalls mit **FFH**.
Der entsprechende Befehl **CWD** (convert word to double word) für die 16-Bit-Version füllt das Erweiterungsregister **DX** mit **0000H** oder **FFFFH**.

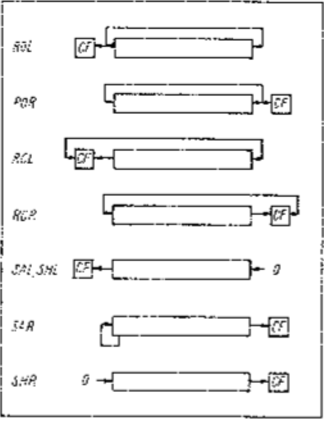


***Bild** 5.2 Wirkung der Rotations- und Shiftbefehle*

5.3.4 *Logikbefehle*

Acht Formen von *Verschiebebefehlen für Rotation und Shift* gestatten Links- und Rechtsverschiebungen von 8- und 16-Bit-Daten in den Registern und im Speicher. Alle Verschiebebefehle können einfach oder mehrfach ausgeführt werden, im zweiten Fall steht im Register **CL** die Zahl der Verschiebungen.
Die *Rotationsbefehle* **ROL** (rotate left) und **ROR** (rotate right) führen zur Links- oder Rechtsrotation des vorgegebenen Operanden. Das auslaufende Bit wird zugleich in das **C**-Flag gesetzt.
Die Befehle **RCL** (rotate through carry flag left) und **RCR** (rotate through carry flag right) schließen das **C**-Flag in die Rotation ein.

Beispiele:
```
RCL AX,1        ;1 × Linksrotation
                ;in <AX>
RCR DX,CL       ;Anzahl der
                ;Rotationen in <CL>
ROL WORD PTR [SI],1
;1 × Linksrotation des Wortes in
;Adresse DS:<SI>
```

Im Fall der Einzelbitrotationen wird das **O**-Flag als weiteres Flag gesetzt. Für **ROL** und **RCL** ergibt sich **OF** aus einer Antivalenzoperation des werthöchsten Bit des Operanden und des C-Flags, während sich die Antivalenzoperation bei den Befehlen **ROR** und **RCR** auf die zwei werthöchsten Bit des Operanden bezieht.
Die *Shift-Befehle* **SHL** (shift logical left) für die logische Links- und **SHR** (shift logical right) für die logische Rechtsverschiebung unterscheiden sich von den Rotationsbefehlen durch das Nachschieben von '**0**'-Stellen. Die auslaufende Bitstelle wird nach jedem Zyklus in das **C**-Flag gesetzt.
Arithmetische Shiftbefehle **SAL** (shift arithmetic left) und **SAR** (shift arithmetic right) verarbeiten vorzeichenbewertete Zahlen. Die Verschiebung mit dem Einschieben von '**0**' betrifft nur den Betragsanteil, ohne die werthöchste, das Vorzeichen enthaltende Bitstelle zu verändern.
Die Wirkung der Rotations- und Shift-Befehle faßt Bild 5.2 zusammen.
Die Beeinflussung des **O**-Flags ist auch bei

den Shift-Befehlen auf die Einzelbitverschiebung beschränkt. Dabei gelten für die Befehle **SHL** und **SAL** die gleichen Bedingungen wie bei **ROL**, **RCL** und für **SHR** wie bei den Befehlen **ROR**, **RCR**. Abweichend davon wird nach dem Befehl **SAR** für die Einzelbitverschiebung **OF** = '0' gesetzt.

Bei den Shift-Befehlen werden zusätzlich die Flags **PF**, **SF** und **ZF** entsprechend den im Abschnitt 5.3.3 angegebenen Bedingungen gebildet.

Der Befehl **NOT** realisiert das Einerkomplement des Operanden ohne Flagbeeinflussung.



Die Befehle **AND**, **OR** und **XOR** (Antivalenz, exclusive or) führen bitweise logische Verknüpfungen zwischen zwei Operanden aus. Als Operanden sind die für die arithmetischen Befehle angegebenen Kombinationen von Registern, Speicherplätzen und im Befehl angegebenen Konstanten möglich.



Die Flags **PF**, **SF** und **ZF** werden in Übereinstimmung mit den Bildungsregeln und **CF** = **OF** = '0' gesetzt.

Der Befehl **TEST** realisiert eine **AND**-Verknüpfung ohne Ergebnisdarstellung, wobei nur die Flags beeinflußt werden.



5.3.5 Befehle zur Programmablaufsteuerung

Sprungbefehle verändern den Inhalt des Befehlszählers IP und im Falle von Sprüngen zwischen verschiedenen Codesegmenten auch den des Codesegmentregisters.

Bei *unbedingten* Sprüngen werden drei Arten unterschieden:

1. Der Befehl **JMP** für Sprünge *innerhalb* eines Codesegmentes existiert mit direkter und indirekter Adressierung.

Eine *direkt* angegebene Adresse gibt den vorzeichenbehafteten 2-Byte-Abstand gegenüber dem aktuellen Befehlszähler an.



2. Bei der *indirekten* Adressierung kann die Ziel-Offsetadresse in einem der acht 16-Bit-Register oder auf zwei benachbarten Speicherplätzen stehen.

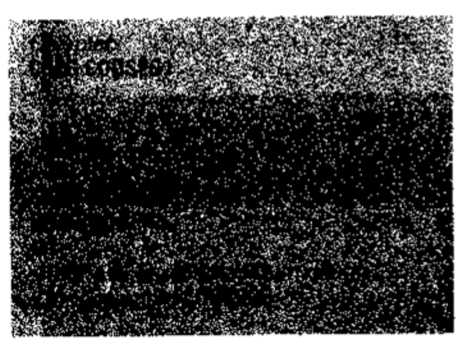


3. Die Kurzform für die direkte Adressierung **JMPS** enthält im Befehl einen vorzeichenbehafteten 1-Byte-Relativabstand.

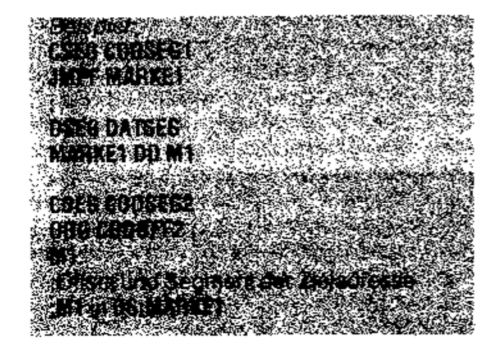


Intersegmentsprünge zu einem neuen Codesegment mit der Bezeichnung **JMPF** (jump far) müssen zusätzlich die neue Codesegmentadresse enthalten.

Dabei wird für eine *direkte* Adressierung die Zieladresse in der Assemblernotierung einfach in einem anderen Codesegment definiert.



Bei Intersegmentsprüngen mit *indirekter* Adressierung wird die Zieladresse mit je zwei Byte für Offset- und Segmentadresse mit der DD-Anweisung im Speicher abgelegt.



Für *bedingte* Sprünge gibt es nur die Kurzform der direkten Adressierung mit 1-Byte-Relativabstand. Die Ausführung des Sprunges ist von Einzelflags oder Flagkombinationen abhängig. Die Assemblerprogramme lassen für einige Befehle zwei mnemonische Beschreibungen zu:

Name	Sprungbedingung
JZ (zero)	<**ZF**> = 1
= **JE** (equal)	
JNZ (not zero)	<**ZF**> = 0
= **JNE** (not equal)	
JB (below)	<**CF**> = 1
= **JNAE** (not above or equal)	
JNB (not below)	<**CF**> = 0
= **JAE** (above or equal)	
JS (sign)	<**SF**> = 1
JNS (not sign)	<**SF**> = 0
JP (parity)	<**PF**> = 1
= **JPE** (parity even)	
JNP (not parity)	<**PF**> = 0
= **JPO** (party odd)	
JO (overflow)	<**OF**> = 1
JNO (not overflow)	<**OF**> = 0
JBE (below or equal)	<**ZF**> = 1 oder
= **JNA** (not above)	<**CF**> = 1
JNBE (not below or equal)	<**ZF**> = 0 und
= **JA** (above)	<**CF**> = 0
JL (less)	<**SF**> ungleich
= **JNGE** (not greater or equal)	<**OF**>
JNL (not less)	<**SF**> gleich
= **JGE** (graeter or equal)	<**OF**>
JLE (less or equal)	<**SF**> ungleich <**OF**>
= **JNG** (not greater)	oder <**ZF**> = 1
JNLE (not less or equal)	<**SF**> gleich <**OF**>
= **JG** (greater)	und <**ZF**> = 0

Eine Sonderform des bedingten Sprunges stellt der Befehl **JCXZ** dar, der nur unter der Bedingung <CX> = 0 ausgeführt wird.

Die *Schleifenbefehle* realisieren eine zyklische Ausführung von Sprungbefehlen. Im Befehl **LOOP** dient das Register CX als Schleifenzähler. Der Sprung wird ausgeführt, solange der Inhalt von **CX** größer Null ist. In jedem Zyklus wird der Inhalt von CX um 1 vermindert.



In weiteren Formen von Schleifenbefehlen ist die Ausführung des Sprunges zusätzlich vom Z-Flag abhängig.

Name	Sprungbedingung
LOOPZ	;<**CX**> > 0 und <**ZF**> = 1
= **LOOPE**	
LOOPNZ	;<**CX**> > 0 und <**ZF**> = 0
= **LOOPNE**	

CALL-Befehle zum Aufruf von Unterprogrammen unterscheiden ebenfalls wie bei **JUMP**-Befehlen Sprünge innerhalb eines Codesegments bzw. zwischen verschiedenen Codesegmenten.

Ein Unterprogrammaufruf *innerhalb* eines Segmentes mit dem Befehl **CALL** enthält wie der entsprechende **JMP**-Befehl nur Angaben zum Offset der Zieladresse, da das Codeseg-

ment nicht verändert wird. Bei der Form mit *direkter* Adressierung wird der vorzeichenbehaftete 2-Byte-Abstand angegeben. Bei der *indirekten* Adressierung steht die absolute Offsetadresse in einem 16-Bit-Operandenregister oder auf 2-Byte-Speicherplätzen mit den beim **JUMP**-Befehl erläuterten Adressierungsformen.
Vor der Ausführung des Unterprogrammaufrufs innerhalb eines Segmentes wird die Offsetadresse des dem **CALL**-Befehl folgenden Befehles automatisch im Stack abgespeichert.
Bedingte **CALL**-Befehle sind im System **8086** nicht vorhanden.

Der Befehl **CALLF** (call far) führt zu Unterprogrammaufrufen zwischen den Codesegmenten. Eine direkt oder indirekt angegebene Zieladresse muß wie beim **JMPF**-Befehl den Segment- und Offsetanteil enthalten.
Bei einem **CALL-FAR** wird die aktuelle Rückkehradresse, bestehend aus Codesegment und Instruction-Pointer, automatisch in 4 Byte im Stack abgelegt. Aus den unterschiedlichen Abspeicherungsformen der Rückkehradresse im Stack ergeben sich unterschiedliche *RETURN*-Befehle. Der Befehl **RET** für den Abschluß von mit **CALL** aufgerufenen Unterprogrammen überträgt nur 2 Byte aus dem Stack in den Befehlszähler **IP**.

Mit **CALLF** angewählte Unterprogramme müssen mit **RETF** abgeschlossen werden, da dieser Befehl neben dem Instruction-Pointer auch das Codesegment aus dem Stack liest. Beide Formen der **RETURN**-Befehle lassen die Angabe von positiven 8- oder 16-Bit-Konstanten zu. Diese geben eine an die **RETURN**-Operation anschließende Erhöhung des SP-Registers an.

Tafel 5.1 Beispiele für 8086-Befehle

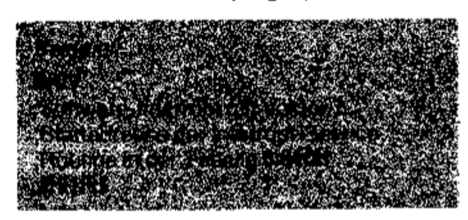
```
                    ;**************************************
                    ;BEISPIELE FUER 8086-BEFEHLE
                    ;**************************************
  1000              CODSEG1  EQU     1000H     ;CODESEGMENT
  3000              CODSEG2  EQU     3000H     ;CODESEGMENT
  5000              ISRSEG   EQU     5000H     ;CODESEGMENT
  2000              DATSEG   EQU     2000H     ;DATENSEGMENT
  4000              EXSEG    EQU     4000H     ;EXTRASEGMENT
  0000              STCKSEG  EQU     0000H     ;STACKSEGMENT
  0000              INTSEG   EQU     0000H     ;INT.-TAB.-SEGM.
  0600              STCKP    EQU     0600H     ;STACKPOINTER
  0010              INTTAD   EQU     0010H     ;INT.-TAB.OFFSET
  0356              ISROFF   EQU     0356H     ;ISR-OFFSET
  6A39              CODOFF2  EQU     6A39H     ;COD.-SEGM.2 OFFS.
  15D0              HALTOFF  EQU     15D0H     ;COD.-SEGM.1 OFFS.
  0021              COUNT    EQU     0021H     ;ZAEHLER
  57E6              ADR      EQU     57E6H     ;KONSTANTE
                    ;
  1000                       CSEG    CODSEG1
                             ORG     0
                    ;TRANSPORTBEFEHLE
0000 8AC6                    MOV     AL,DH
0002 8BD5                    MOV     DX,SI
0004 B8086B                  MOV     AX,6B08H
0007 B9E657                  MOV     CX,ADR
000A 890E0000                MOV     NAME,CX
000E 8B0E0200                MOV     CX,MARKE
0012 2E8B0ED215              MOV     CX,ADR2
0017 BA230F                  MOV     DX,0F23H
001A 8B163DF2                MOV     DX,.0F23DH
001E 8B77FE         ADR1:    MOV     SI,[BX]-2
0021 887308                  MOV     [BP+DI]+8,BH
0024 368904                  MOV     SS:[SI],AX
0027 BE423E                  MOV     SI,3E42H
002A B564                    MOV     CH,100
002C BB1E00                  MOV     BX,OFFSET ADR1
002F C74403FF37              MOV     WORD PTR [SI]+3,37FFH
0034 C60101                  MOV     BYTE PTR [BX+DI],1
0037 8D98670A                LEA     BX,[BX+SI]+0A67H
003B C50B                    LDS     CX,[BP+DI]
003D D7                      XLAT    AL
003E 8735                    XCHG    SI,[DI]
                    ;LADEN DER SEGMENTE
0040 B80000                  MOV     AX,STCKSEG
0043 8ED0                    MOV     SS,AX
0045 B80000                  MOV     AX,INTSEG
0048 8ED8                    MOV     DS,AX
004A B80040                  MOV     AX,EXSEG
004D 8EC0                    MOV     ES,AX
                    ;LADEN STACKPOINTER, INTERRUPT-TABELLE
004F BC0006                  MOV     SP,STCKP
0052 C7061C005603            MOV     TABOFF,OFFSET ISR
0058 C7061E000050            MOV     TABSEG,ISRSEG
005E B80020                  MOV     AX,DATSEG
0061 8ED8                    MOV     DS,AX
                    ;E/A-BEFEHLE
0063 E480                    IN      AL,80H
0065 EF                      OUT     DX,AX
                    ;FLAG-BEFEHLE
0066 9F                      LAHF
0067 9E                      SAHF
                    ;MOV-STRING-BEFEHL
0068 FC                      CLD
0069 BE1A00                  MOV     SI,OFFSET SRCPTR
006C BF1200                  MOV     DI,OFFSET DSTPTR
006F B91000                  MOV     CX,16
0072 F3A4                    REP     MOVS BYTE PTR [DI],[SI]
                    ;COMPARE-STRING-BEFEHL
0074 BE0800                  MOV     SI,OFFSET TB1
0077 BF0000                  MOV     DI,OFFSET TB2
007A B96400                  MOV     CX,100
007D F3A6                    REP     CMPS TB2,TB1
                    ;STORE-STRING-BEFEHL
007F BF2200                  MOV     DI,OFFSET TAB
0082 B90004                  MOV     CX,400H
0085 B020                    MOV     AL,' '
0087 F3AA                    REP     STOS BYTE PTR [DI]
                    ;LOAD-STRING-BEFEHL
0089 BE0800                  MOV     SI,OFFSET TB1
008C AC                      LODS    BYTE PTR [SI]
                    ;SCAN-STRING-BEFEHL
008D B043                    MOV     AL,'C'
008F BF0000                  MOV     DI,OFFSET TB2
0092 B9FFFF                  MOV     CX,0FFFFH
0095 F2AE                    REPNZ   SCAS BYTE PTR [DI]
                    ;ARITHMETIK-LOGIK-VERSCHIEBE-BEFEHLE
```

```
0097 010C                       ADD     [SI],CX
0099 2E803D030041               CMP     CS:BYTE PTR [DI]+3,'A'
009F FF07                       INC     WORD PTR [BX]
00A1 F6E7                       MUL     BH
00A3 3EF66600                   IMUL    DS:BYTE PTR [BP]
00A7 F7F3                       DIV     BX
00A9 D1D0                       RCL     AX,1
00AB D3CB                       ROR     BX,CL
00AD D104                       ROL     WORD PTR [SI],1
00AF F610                       NOT     BYTE PTR [BX+SI]
00B1 802752                     AND     BYTE PTR [BX],52H
00B4 234D07                     AND     CX,[DI]+7
00B7 A880                       TEST    AL,80H
00B9 13C3                       ADC     AX,BX
                       ;SOFTWARE-INTERRUPT
00BB CD07                       INT     7
                       ;JUMP-BEFEHLE
00BD E93112      12F1           JMP     $+1234H
00C0 E95DFF      001E           JMP     ADR1
00C3 FFE3                       JMP     BX
00C5 FF6408                     JMP     WORD PTR [SI]+8
00C8 EB09        00D3           JMPS    ADR3
00CA EA396A0030                 JMPF    M1
00CF FF260200                   JMP     MARKE
00D3 FF2E0400          ADR3:    JMPF    MARKE1
00D7 8B2100                     MOV     CX,COUNT

00DA 90                ADR4:    NOP
00DB E2FD        00DA           LOOP    ADR4
                       ;CALL-BEFEHLE
00DD E81000      00F0           CALL    ADR5
00E0 FF160200                   CALL    MARKE
00E4 9A396A0030                 CALLF   M1
00E9 FF1E0400                   CALLF   MARKE1
00ED E80000      00F0           CALL    $+0003
00F0 C3                ADR5:    RET
                       ;
                                ORG     HALTOFF
                       ;CPU-STEUERBEFEHLE
15D0 F4                         HLT
15D1 9B                         WAIT
                       ;
15D2 7803              ADR2     DW      0378H
                       ;
                       ;INTERRUPT-SERVICE-ROUTINE FUER INT7
  5000                          CSEG    ISRSEG
                                ORG     ISROFF
0356 1E                ISR:     PUSH    DS
0357 9C                         PUSHF
0358 9D                         POPF
0359 1F                         POP     DS
035A CF                         IRET
                       ;
                       ;
  3000                          CSEG    CODSEG2
                                ORG     CODOFF2
6A39 CB                M1:      RETF
                       ;
  0000                          DSEG    INTSEG
                                ORG     INTTAB
001C                   TABOFF   RW      1
001E                   TABSEG   RW      1
                       ;
  2000                          DSEG    DATSEG
                                ORG     0
0000                   NAME     RW      1
0002 D015              MARKE    DW      HALTOFF
0004 396A0030          MARKE1   DD      M1
0008 5A4549434845      TB1      DB      'ZEICHENKETTE IN DS'
     4E4B45545445
     20494D204453
001A 444153204953      SRCPTR   DB      'DAS IST EIN TEST'
     542045494E20
     54455354
                       ;
  4000                          ESEG    EXSEG
                                ORG     0
0000 5A4549434845      TB2      DB      'ZEICHENKETTE IN ES'
     4E4B45545445
     20494D204553
0012                   DSTPTR   RS      16
0022                   TAB      RS      400H
                                END

END OF ASSEMBLY. NUMBER OF ERRORS: 0. USE FACTOR: 12
```

Mit dem speziellen Befehl **INT** (softwareinterrupt) kann direkt eine Interrupt-Service-Routine in einem beliebigen Codesegment aufgerufen werden. Dabei ist im Befehl nur die Angabe des Interrupt-Vektors erforderlich. Die Startadresse der Interrupt-Service-Routine wird dann nach Dekodierung des **INT**-Befehls aus den entsprechenden Plätzen in der Interrupt-Tabelle gelesen.
Vor dem Start der Interrupt-Service-Routine werden das Flagregister und die vollständige Absolutadresse mit Segment und Offset des dem **INT**-Befehl nachfolgenden Befehls automatisch im Stack abgespeichert.
Das *I*-Flag und das *T*-Flag werden gelöscht, so daß weitere Interrupts gesperrt sind.



Der Befehl **INTO** ist ein Spezialfall des INT-Befehls für den interrupt-Vektor 4. Die Ausführung ist von der Bedingung 0-Flag = 1 abhängig.
Die durch Hardware-Interrupt oder Software-Interrupt ausgelösten Routinen müssen durch einen speziellen Rückkehrbefehl **IRET** (interrupt return) abgeschlossen werden.
Der Befehl **IRET** überträgt die im Stack abgelegten Informationen zum Zustand vor dem Interrupt in den Befehlszähler, in das Codesegment und in die Flags. Damit wird die Interruptfähigkeit auch ohne zusätzliche Befehlsnotierung wieder hergestellt.

5.3.6 Befehle für Steuerfunktionen

Mit dem Befehl **WAIT** kann die Programmbearbeitung gestoppt werden. Während der Ausführung dieses Befehls wird der **TEST**-Eingang der **8086-CPU** abgefragt. Dabei verbleibt der Prozessor in einem Wartezustand, solange an diesem Anschluß High-Pegel anliegt.

Quellprogramm (ED, WS, TP)
Assemblierung (ASM 86)
Syntaxfehler ?
Quelltextkorrektur (ED, WS, TP)
Erzeugen eines CMD-Files (GENCMD)
Programmlauf (SCPX)
Fehler ?
Fehlersuche, Programmtestung (DDT86, SID 86)
Assemblierung (RASM 86)
Syntaxfehler ?
Quelltextkorrektur (ED, WS, TP)
Linken (LINK 86)
Programmlauf (SCPX)
Fehler ?
Fehlersuche, Programmtestung (SID 86)
bearbeitungsfähiges fehlerfreies Programm

Bild 6.1 Softwareentwicklungszyklus

Der Befehl **HLT** bringt den Prozessor in den Halt-Zustand, der nur durch einen Interrupt wieder aufgehoben werden kann. Durch das 1-Byte-Präfix **LOCK** vor einem Befehl wird der Ausgang '**LOCK**' des Prozessors für die Dauer der Befehlsausführung aktiv. Durch schaltungstechnische Maßnahmen wird in Mehrprozessorsystemen gesichert, daß der **CPU** während eines über mehrere **BUS**-Zyklen reichenden Befehls der **BUS**-Zugriff nicht verlorengeht.
Befehle, die mit dem 5-Bit-Präfix **ESC** (Codierung **11011**——) beginnen, sind für die Zusammenarbeit mit den Koprozessoren des Systems 8086 vorgesehen.
Die **8086-CPU** führt für diese Befehle auch einige Funktionen aus. Die im Befehl angegebene Speicheradresse wird von der **CPU** als physische 20-Bit-Adresse generiert und auf den Adreßbus gelegt. Danach wird durch Steuersignale zwischen den verschiedenen Prozessoren die Kontrolle an einen Koprozessor übergeben, welcher von der selektierten Adresse den Dateninhalt übernimmt. Nach dieser Übernahme werden vom Koprozessor gegebenenfalls benötigte Folgeadressen für weitere Speicherzugriffe eigenständig generiert.
Genauere Darlegungen erfolgen im Abschnitt Koprozessoren.
Beispiele für die **8086**-Befehle bei einem mit **ASM86** assemblierten Quelltext sind in Tafel 5.1 dargestellt.

6. Assemblerprogrammierung mit dem Betriebssystem SCP 1700

6.1 Systemprogramme zur Assemblerprogrammierung

Das Betriebssystem **SCP 1700** enthält eine Reihe von Softwarekomponenten, welche die Assemblerprogrammierung unterstützten. Man kann den Entwicklungsvorgang in 3 Phasen einteilen:

1. Phase: Editieren der Programme
Hilfsmittel:
- Editor **ED**
- Textverarbeitungsprogramme **WS** und **TP**

2. Phase: Assemblieren und Maschinencodeerzeugung
Hilfsmittel:
- Absolutassembler **ASM86**
- Relativassembler **RASM86**
- Filegenerierprogramm **GENCMD**
- Linker **LINK86**

3. Phase: Testung und Fehlersuche
Hilfsmittel:
- Debugger **DDT86**
- symbolischer Debugger **SID86**

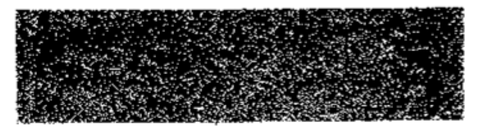
Folgende Hilfsmittel sind als Minimalausstattung anzusehen:
- **ED**
- **ASM86**
- **GENCMD**
- **DDT86** oder **SID86**.

Unter Nutzung der 8-Bit-Technik gibt es direkte Äquivalente:
- **ED**
- **ASM86**
- **GENCMD**,

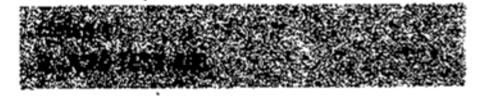
so daß Vorarbeiten auf der Basis der Crosssoftwareentwicklung, die im System **SCP 1700** oder zu dessen Implementierung genutzt werden können, möglich sind. In Bild 6.1 ist übersichtsmäßig der Entwicklungsweg von der Problemlösung bis zum fertigen Programm dargelegt.

6.2 Der Zeileneditor ED

Der Zeileneditor **ED** läßt sich im Betriebssystem **SCP 1700** universell einsetzen, so z. B. zum Editieren von Assemblerquelltexten, von Hochsprachprogrammen (z. B. **BASIC**, **PASCAL**, **C**) und Texten allgemeiner Art.
Für die Arbeit mit **ED** empfiehlt es sich vor Arbeitsbeginn, das System **SCP 1700** auf das aktuelle Arbeitslaufwerk einzustellen.



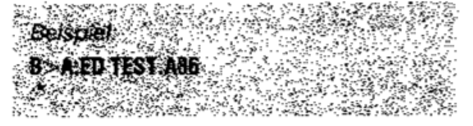
Hierbei können weitgehend Fehler vermieden werden, die durch das ungewollte Beschreiben der Systemdiskette o. ä. entstehen können. Weiterhin erhöht sich die Arbeitsgeschwindigkeit der Dienstprogramme bei der Bearbeitung der Areitsdateien. Zum Starten von **ED** muß das Kommandowort "**ED**" **mit** einer Dateispezifikation **ohne** Sonderzeichen eingegeben werden.



Ein Vergessen der Angabe der Dateispezifikation kann zu unkontrollierten Schreibzugriffen auf der Diskette führen, die unter Umständen das Directory zerstören können! Für besondere Fälle ist es erforderlich, die neue Datei unter einem anderen Namen und eventuell auf einem anderen Diskettenlaufwerk ablegen zu müssen. Die Kommandoeingabe sieht dann wie folgt aus:

Beispiel:
B>A:ED TEST.A86 d:TEST1.A86

Dabei gilt die Voraussetzung, daß die zweite Datei noch nicht existent ist! ED meldet sich nach dem Start mit seinem Kommandoprompt und erwartet eine Kommandoeingabe:



Literatur
/1/ Rextor, R.; Alexy, G.: Das 8086/8088 Buch, Programmieren in Assembler und Systemarchitektur. te-wi Verlag, München 1982
/2/ Jorke, G.; Lampe, B.; Wengel, N.: Arithmetische Algorithmen der Mikrorechentechnik. VEB Verlag Technik, Berlin 1989
/3/ Intel – Applikation Note AP-59 Using the 8259A Programmable Interrupt Controller. September 1979

wird fortgesetzt

Mikroprozessorsystem K 1810 WM 86

Hardware · Software · Applikation (Teil 4)

Prof. Dr. Bernd-Georg Münzer
(wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann,
Wolfgang Kabatzke, Frank Kamrad,
Helfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock,
Sektion Technische Elektronik,
Wissenschaftsbereich Mikrorechentechnik/
Schaltungstechnik

Für den Fall, daß die Datei eine neu zu erstellende Datei ist, wird folgende Mitteilung mit ausgegeben:



ED erlaubt eine Reihe von Kommandos. Die *wichtigsten* sollen nachfolgend erläutert werden.

6.2.1 Texttransferkommandos

Die möglichen Kommandos sind:

nA *Append*
Anfügen von n unbearbeiteten Quellzeilen aus der Quelldatei an den gefüllten Teil des Arbeitspuffers. Wird für nA ein **#A** eingegeben, werden zirka 48 KByte als Textpuffer reserviert. Kleine Dateien können mit #A zum Beginn des Editiervorganges komplett geladen werden. Wird für nA ein **0A** eingegeben, werden nur so viel Textzeilen eingelesen, bis der Arbeitspuffer halb gefüllt ist. Wird kein n eingegeben, liest ED nur eine Quellzeile ein.

nW *Write*
Schreiben der ersten n Zeilen zur temporären Hilfsdatei. Diese Zeilen werden in der neuen Datei an das Ende der schon eingetragenen Zeilen angefügt. Wird für nW ein **0W** eingegeben, werden soviel Zeilen in die neue Datei geschrieben, bis der Arbeitspuffer etwa halb leer ist. Nach der Ausführung eines W-Kommandos muß das **H**-Kommando gegeben werden, wenn erneut auf die geretteten Zeilen zugegriffen werden soll.

E *Exit*
Beenden von ED. Es werden die gesamten gepufferten Zeilen und die unbearbeiteten Zeilen zur neuen Datei kopiert. Dabei wird die alte Datei in eine Datei mit dem Dateisuffix .BAK umgewandelt.

6.2.2 Editiergrundkommandos

B *Begin*
–B
Bewegen des Characterpointers (**CP**) an den Beginn (**B**) oder das Ende (**–B**) des Arbeitspuffers.

nC
–nC
Bewegen des **CP** um n Zeichen vorwärts oder rückwärts.

nD *Delete*
–nD
Löschen von n Zeichen vor (**nD**) oder hinter (**–nD**) dem CP.

nK *Kill*
–nK
Löschen von n Zeilen vor dem **CP** (**nK**) oder hinter dem **CP** (**–nK**).

nT *Type*
–nT
Anzeige von n Zeilen vor dem **CP** (**–nT**) oder hinter dem **CP** (**nT**) auf dem Terminal.

n
–n
Bewegen des CP um n Zeilen vor den **CP** (**–n**) oder hinter den **CP** (**n**). Ein Betätigen von **CR** (**ENTER**) wird wie die Eingabe des Kommandos

:1

interpretiert, d. h., der **CP** wird jeweils um eine Zeile weiter gestellt.

I *Input*
i
Einstellen des Eingabemodus. Es werden solange Textzeilen **vor** den **CP** eingegeben, bis die Eingabe von **CTRL–Z** erfolgt ist.

6.2.3 Kommandos zur Modifizierung des Arbeitspuffers

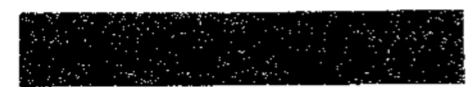
S *Substitute*
s
Ersetzen von Zeichen und Zeichenketten im Arbeitspuffer. Das Kommando wird wie folgt gegeben:
nssearch_string^znew_string
wobei n die Anzahl der Substitutionen angibt.



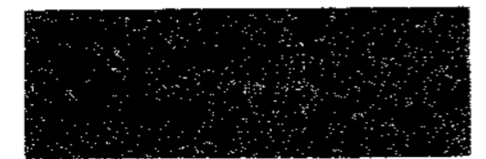
Es erfolgt keine Anzeige der geänderten Zeile. Soll diese mit angezeigt werden, muß eine Kommandoverkettung in folgender Weise eingegeben werden:



Mit Hilfe des S-Kommandos lassen sich auch Zeichen und Zeichenketten streichen.



Weiterhin ist eine Kommandoverkettung auf der Basis der **C-**, **D-**, **L-**, **K-** und **T**-Kommandos möglich.





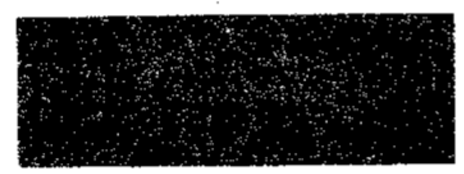
6.2.4 Fortgeschrittene ED-Kommandos

nP Page
–nP
Dieses Kommando bewirkt eine komplette bildschirmfüllende Anzeige und bewegt den CP an den Anfang einer neuen Bildschirmseite. Wenn keine Seitenzahl angegeben wurde, wird der CP um 23 Zeilen vorwärts bewegt und die weiteren 23 folgenden Zeilen werden angezeigt. Mit 0P kann die aktuelle Seite angezeigt werden, ohne daß der CP weiter bewegt wird.

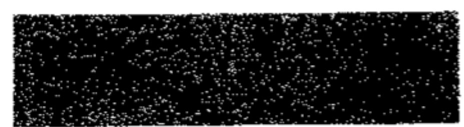
n: Line Number
Mit Hilfe dieses Kommandos kann der CP auf eine spezifizierte Zeile eingestellt werden.



:n Trogh Line Number
Das zum Line-Number-Kommando inverse Kommando wird bis zu einer bestimmten Zeilennummer ausgeführt.



4:*
nFstring{^z}
Suchfunktion, wobei n eine Zahl ist, die das n-te Auftreten der gesuchten Zeichenkette angibt. Wichtig hierbei ist, daß die Zahl n eine positive Zahl ist, da **ED** im Puffer nur *vorwärts* suchen kann.



Der CP wird auf die Zeile *nach* der gesuchten Zeichenkette positioniert, wo (in diesem Beispiel) das dritte Auftreten der Zeichenkette zyklisch lokalisiert wurde. Durch die Kombination des **F**- mit dem **T**-Kommando kann die Zeile mit angezeigt werden:



Wenn ED die Zeichenkette im Arbeitspuffer

nicht findet, wird eine Fehlermeldung ausgegeben (siehe 6.2.7).

nNstring{^z}

Das N-Kommando sucht über den Arbeitspuffer hinaus in der *gesamten* Quelle. Der Zeichenkette muß ein CTRL-Z folgen, wenn noch ein weiteres Kommando angehängt werden soll. Wenn das Editieren fortgesetzt werden soll, nachdem die Quelldatei bearbeitet und der Arbeitspuffer geleert ist, **muß** das H-Kommando angewendet werden.

6.2.5 Transport von Textblöcken

Diese Kommandos werden benutzt, um eine Anzahl von Textzeilen von einem Dateibereich in einen anderen zu transportieren. Hierzu muß zuerst das X-Kommando benutzt werden, welches den gewünschten Textblock in eine Temporärdatei (XOOOOOOO.LIB) schreibt. Danach werden die Originalzeilen mit dem K-Kommando aus dem Text gelöscht und der Textblock mit dem R-Kommando an das gewünschte Ziel geladen.

nX

nXfilespec

Die Zahl n gibt die Anzahl der Zeilen ab CP in Richtung Pufferende an, die in die Temporärdatei zu übertragen sind (n **muß** positiv sein). Wird kein Dateiname angegeben, nimmt ED automatisch XOOOOOOO.LIB an. Ist kein Dateisuffix spezifiziert, wird automatisch .LIB zugewiesen.

R

Rfilespec

Mit dem R-Kommando können die Zeilen zurück übertragen werden, die mit dem **X**-Kommando in die Temporärdatei geschrieben wurden bzw. es können andere Quelltextdateien in die zu bearbeitende eingelesen werden. Ein nicht angegebenes Dateisuffix wird durch **.LIB** ersetzt. Das **R**-Kommando liest die Datei **vor** den **CP** ein.

6.2.6 Beendigung von ED und Dateisicherung

H *Head of File*

Ein H-Kommando sichert den Inhalt des Arbeitspuffers, ohne jedoch den Editiervorgang zu beenden und kehrt zum Kopf (Head) der Datei zurück. Auf die bereits editierten Daten kann erneut zurückgegriffen werden.

O *Original*

Ein O-Kommando schließt den Editiervorgang ab und läßt die Datei im Originalzustand zum erneuten Editieren, ohne **ED** abzubrechen. ED beantwortet die Eingabe des O-Kommandos mit der Frage:

O(Y/N)?

Diese muß mit Y beantwortet werden, wenn das O-Kommando ausgeführt werden soll. Mit N kehrt man in den normalen ED-Betrieb zurück. Jede andere Taste wiederholt die Frage.

Q *Quit*

Ein Q-Kommando schließt den Editiervorgang ab und beendet ED. Bei der Eingabe des Q-Kommandos wird ebenfalls abgefragt:

Q(Y/N)?

Geantwortet werden muß mit **Y** oder **N**, jedes andere Zeichen wiederholt die Frage. Die Temporärdatei wird gelöscht, die Quelldatei wird geschlossen, und es wird keine Backup-Datei mit dem Dateisuffix .BAK erzeugt.

E *Exit*

(siehe 6.2.1)

6.2.7 ED-Fehlermeldungen

Die allgemeine Form der **ED**-Fehlermeldung ist:

BREAK "x" **AT** c,

wobei x ein Fehlersymbol darstellt und c der Buchstabe des Kommandos ist, bei dem der Fehler aufgetreten ist.

Fehlersymbol	Bedeutung
#	Suchfehler
?c	unbekannter Kommandobuchstabe
O	keine **.LIB**-Datei
>	Puffer voll
E	Kommandoabbruch. Eine Tastatureingabe hat das Kommando abgebrochen
F	Dateifehler. Es erfolgt entweder DISK FULL oder **DIRECTORY FULL.**

Hinweis:

ED interpretiert die Kommandoeingaben je nachdem, ob sie als Groß- oder Kleinbuchstaben erfolgten. Es ist die generelle Eingabe als Kleinbuchstaben zu empfehlen, da hierbei **alle** Buchstaben, ob Klein- oder Großbuchstaben im Text gesucht, gefunden oder bearbeitet werden können. Eine Kommandoeingabe als Großbuchstabe erlaubt nur die Bearbeitung von Texten, die generell aus Großbuchstaben und Ziffern bestehen.

6.3 Der Absolutassembler ASM86 und der Relativassembler RASM86

ASM86 und **RASM86** wurden für die Realisierung verschiedener Aufgaben geschaffen:

ASM86

Assemblierung von absoluten Programmen. Eine modulare Programmierung ist nicht oder nur sehr schlecht möglich.

RASM86

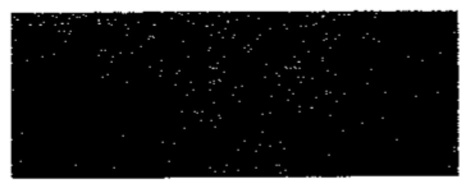
Assemblierung von verschieblichen Programmen und Programmodulen. Eine Anbindung an Programme, die von anderen Übersetzern erzeugt wurden (z.B. **PASCAL, C, FORTRAN77**) ist möglich.

6.3.1 Bedienung

Die Assembler **ASM86** und **RASM86** werden nach einem einheitlichen Konzept bedient. Die Kommandozeile sieht allgemein wie folgt aus:



Es gibt bei den Dateinamen eine Standardannahme in der Weise, daß die Quelldateien mit dem Dateisuffix .A86 versehen sind. In diesem Falle braucht das Dateisuffix beim Assembleraufruf in der Angabe **name** nicht mit angegeben werden. Sollen Dateien mit einem anderen Suffix assembliert werden, so ist dieses mit anzugeben. Nach ihrem Aufruf melden sich die Assembler mit einer Identifikationsausschrift:



wobei die Angabe x.x die jeweils eingesetzte Versionsnummer angibt. Stimmt der Dateiname nicht oder ist er nicht auf dem angegebenen Laufwerk vorhanden, geben beide Assembler die Fehlermeldung

NO FILE

aus und brechen die Arbeit ab. Entsprechend dem jeweiligen Assemblerkonzept können die Assembler aus dem Quelltext verschiedene Ergebnisdateien erzeugen:



Die Angabe der Optionen kann unter Angabe der oben angeführten Parameter und zusätzlicher Geräteparameter die Erzeugung und Abspeicherung der zu erzeugenden Dateien beeinflussen. Als Geräteparameter können die Buchstaben

A, B, . . . P oder **X, Y, Z**

angegeben werden, wobei

A bis P die Laufwerke spezifizieren

X die Bedienkonsole **(CON:)**

Y den Drucker **(LST:)**

Z die Ausgabe unterdrückt **(NUL:)**.

Beim RASM86 ist zusätzlich noch die Option "L" möglich, mit deren Hilfe die lokalen Parameter in die Objektdatei eingeschlossen werden können, welche in der durch **LINK86** erzeugten Symboldatei erscheinen. Andernfalls erscheinen nur **PUBLIC**-Symbole in der Symboldatei. Die erzeugten SYM-Dateien sind für die Arbeit mit dem symbolischen Debugger **SID86** (siehe Abschnitt 6.5) notwendig. Bei der Ausgabeumlenkung auf die Konsole kann die Ausgabe mit **CNTRL-S** angehalten werden und mit **CNTRL-Q** fortgesetzt werden.



dem Laufwerk **C** und erzeugen von TEST.H86 und **TEST.LST. TEST.SYM** wird unterdrückt.

ASM86 TEST ○PY SX

Assemblieren der Datei **TEST.A86** und erzeugen von **TEST.H86.** Die Ausgabe von **TEST.LST** erfolgt direkt auf den Drucker und die Ausgabe von **TEST.SYM** erfolgt direkt auf die Konsole.

RASM86

RASM86 TEST

Assemblieren der Datei **TEST.A86** und erzeugen von **TEST.OBJ, TEST.LST** und **TEST.SYM** auf dem aktuellen Laufwerk.

RASM86 TEST ○AC SZ PZ

Assemblieren von **TEST.A86** auf dem Laufwerk **C** und erzeugen von **TEST.OBJ.** Die Dateien **TEST.SYM** und **TEST.LST** werden unterdrückt.

RASM TEST ○LO

Assemblieren von **TEST.A86** auf dem aktuellen Laufwerk und erzeugen von **TEST.OBJ, TEST.LST** und **TEST.SYM.** Die lokalen Symbole werden in **TEST.OBJ** mit eingeschlossen.

ASM86 und RASM86 können durch Betätigung einer beliebigen Taste angehalten werden. Beide Assembler antworten auf eine Tastenbetätigung mit:



Ein Y bricht den Assemblerlauf ab. Ein N setzt die Arbeit fort.

6.3.2 Elemente der Assemblersprachen für ASM86 und RASM86

Beide Assembler stellen hinsichtlich der Assemblernotation eine Einheit dar. Es gibt keine Unterschiede. Da eine ausführliche Beschreibung am Beispiel des ASM86 in Kapitel 5 erfolgt ist, soll in diesem Kapitel nicht näher darauf eingegangen werden.

6.3.3 Assemblerdirektiven

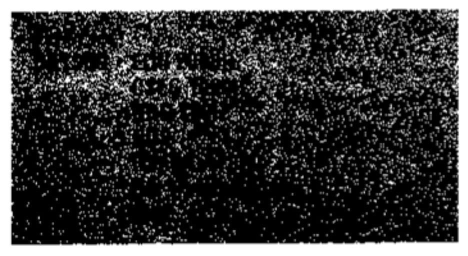
Direktivanweisungen sind wichtig für die Zuweisung von Codeteilen zu logischen Segmenten. Weiterhin lassen sich mit solchen Anweisungen die bedingte Assemblierung, die Datenelementdefinition und die Listenformatsteuerung durchführen. RASM86 enthält im wesentlichen den Direktivanweisungssatz des ASM86, so daß hier nur die wesentlichen Unterschiede bzw. Ergänzungen aufgezeigt werden sollen. Nachfolgend wird eine Übersicht der Direktivanweisungen beider Assembler gegeben.

	RASM86	**ASM86**
CSEG	x	x
DSEG	x	x
SSEG	x	x
ESEG	x	x
ORG	x	x
END	x	x
IF	x	x
ENDIF	x	x
ELSE	x	–
EQU	x	x
DB	x	x
DW	x	x
DD	x	x
RS	x	x
RB	x	x
RW	x	x
RD	x	–
EJECT	x	x
LIST	x	x
NOLIST	x	x
IFLIST	x	–
NOIFLIST	x	–
PAGESIZE	x	x
PAGEWIDTH	x	x
TITLE	x	x
INCLUDE	x	x
SIMFORM	x	x
NAME	x	–
PUBLIC	x	–
EXTRN	x	–
GROUP	x	–

Die geringfügigen Unterschiede bei der Anwendung der Segmentdirektiven und der erweiterte Direktivsatz des RASM86 sollen nachfolgend erläutert werden.

● **Segmentdirektiven**

Die beiden Assembler benötigen die Segmentdirektiven zur Unterscheidung der Code-, Daten-, Stack- und Extrasegmentbereiche. Beim ASM86 können in den Direktivanweisungen numerische Ausdrücke bzw. Operatoren zur Festlegung der Segmentposition angegeben werden.



Die Segmentdirektiven des **RASM86** sind daher dieser Aufgabenstellung mit angepaßt. Segmente können nach folgendem Schema benannt werden:

[seg_name] segment [align_typ]
[combine_typ][class_name]

Als **segment** ist möglich: **CSEG**
DSEG
SSEG
ESEG.

Der Segmentname **seg_name** kann ein beliebiger RASM86-Bezeichner sein, wobei bei Nichtangabe folgende Standardnamen von RASM86 angenommen werden:

Segment-Direktive	Standardnamen
CSEG	**CODE**
DSEG	**DATA**
SSEG	**STACK**
ESEG	**EXTRA**

RASM86 verbindet alle Segmente mit dem gleichen Segmentnamen, auch wenn sie nicht zusammenhängend im Quellcode stehen. Für den Linker **LINK86** kann durch die Angabe des Zuordnungstypes (**align_typ**) eine spezielle Segmentadresse festgelegt werden. Folgende Angaben sind möglich:

BYTE Bytezuweisung
(Segment beginnt ab dem nächsten Byte)
WORD Wortzuweisung
(Segment beginnt an einer geraden Adresse)
PARA Paragraphenzuweisung
(Segment beginnt an einer Paragraphenadresse [A0 – A3 = 0])
PAGE Seitenzuweisung
(Segment beginnt an einer Seitengrenze [A0 – A7 = 0])

Wird kein Typ explizit angegeben, wird folgende Standardtypenzuordnung wirksam:

Segment-Direktive	Zuordnungstyp
CSEG	**BYTE**
DSEG	**WORD**
SSEG	**WORD**
ESEG	**WORD**

Die Angabe des Verbindungstypes (**combine_typ**) ist für den Linker wichtig, damit er Segmente mit anderen Segmenten gleichen Namens verbinden kann. Folgende Verbindungstypen sind angebbar:

PUBLIC
Alle Segmente dieses Verbindungstypes werden in der Reihenfolge ihres Auftretens durch den Linker verkettet. (Standardtyp, wenn keine Angabe erfolgt).

COMMON
Das Segment hat gleiche Speicherplätze mit anderen Segmenten gleichen Namens

STACK
Alle Segmente eines Verbindungstypes werden zu höheren Adressen hin überlagert, weil Stacks abwärts zu niederen Adressen hin wachsen.

LOCAL
Das Segment ist lokal im übersetzten Programm. Es wird nicht mit anderen Segmenten verbunden.

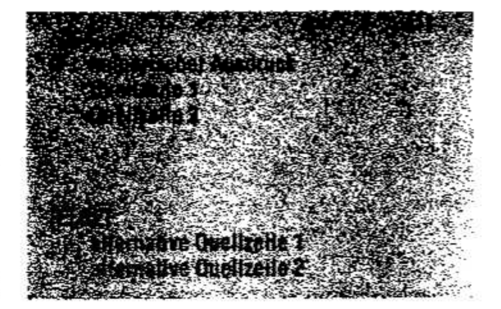
nnnn
Für absolute Programme bestimmt RASM86 die Ladeadresse während des Assemblerlaufes. Die Position wird durch den Linker zum Ladezeitpunkt bestimmt.

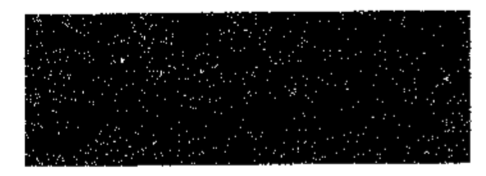
Durch die Angabe des Klassennamens **class_name** können gekennzeichnete Segmente im gleichen Bereich einer vom Linker erzeugten CMD-Datei eingeordnet werden. Falls nicht durch die GROUP-Direktive oder durch Linkerkommandos eine Umstellung erfolgt, ordnet RASM86 die Segmente in die CMD-Datei wie folgt ein:

Segment-Direktive	Standard-Klassenname	CMD-Bereich
CSEG	**CODE**	**CODE**
DSEG	**DATA**	**DATA**
SSEG	**STACK**	**STACK**
ESEG	**EXTRA**	**EXTRA.**

● **ELSE-Direktive**

Im Gegensatz zum ASM86 kann eine alternativ zu übersetzende Folge von Quelltextzeilen angegeben werden, welche übersetzt wird, falls die angegebene Bedingung unwahr ist. Es ergibt sich eine einfachere Bedienung als beim ASM 86.





Die nicht übersetzten Quellzeilen werden aber mit in die Listingdatei aufgenommen, d. h. nur gedruckt, sofern diese nicht unterdrückt werden.

● **RD-Direktive**

[symbol] RD numerischer Ausdruck

Die RD-Direktive reserviert ein Doppelwort im Speicher ohne Initialisierung.

● **IFLIST-/NOIFLIST-Direktive**

Die NOIFLIST-Direktive unterdrückt die Ausgabe des während der bedingten Assemblierung zu übersetzenden Blockes, wenn *keine* Übersetzung angewiesen wurde. Die Ausgabe kann durch die IFLIST-Direktive angewiesen werden.

● **GROUP-Direktive**

group_name **GROUP** seg_name **1**, . . . ,seg_name n

Die **GROUP**-Direktive weist **RASM86** an, die aufgeführten Segmente in eine Gruppe zu verbinden. Die Reihenfolge der Namen bestimmt, in welcher Reihenfolge der Linker die Segmente in die CMD-Datei legt.

● **NAME-Direktive**

NAME 'modul_name'

Die **NAME**-Direktive weist dem von RASM86 erzeugten Objektmodul einen Namen zu. Sofern keine Angabe erfolgt, wird der Name der Quelldatei als Name des Objektmoduls übernommen.

● **PUBLIC-Direktive**

PUBLIC name,{name, . . .}

Die **PUBLIC**-Direktive legt fest, daß mit **PUBLIC** definierte Namen zu anderen miteinander verbundenen Programmen Bezug nehmen können.

● **EXTRN-Direktive**

EXTRN external_id[,external_id, . . .]

Diese Direktive ermöglicht RASM86 den Zugriff innerhalb des übersetzten Programmes zu *jedem* externen Symbol, welches in einem anderen Programm definiert ist. Der external_id besteht aus einem Symbol und einem Typ. Das Symbol kann eine Zahl oder Variable sein. Als Typ sind möglich:



6.4 Das Filegenerierprogramm GENCMD und der Linker LINK86

6.4.1 GENCMD

Mit Hilfe des Programmes GENCMD können Dateien im H86-Format (entstanden z. B. in einem Assemblerlauf mit **ASM86**) in CMD-Dateien, die auf dem Betriebssystem SCP **1700** abarbeitungsfähig sind, umgesetzt werden. **H86**-Dateien können auch vor der Bearbeitung durch **GENCMD** mit Hilfe des Programmes **PIP** zu größeren **H86**-Dateien verkettet werden. Es muß hierbei beachtet werden, daß die einzelnen **H86**-Dateien, die zusammengefaßt werden sollen, Speicherbereiche überlagern können. Bei einer Speicherinhaltsdefinition in einer **H86**-Datei ist für **GENCMD** die *letzte* Definition gültig. Diese Eigenschaft wird beispielsweise bei der Implementation des Betriebssystemes **SCP 1700** ausgenutzt. **GENCMD** wird nach folgendem Schema aufgerufen:

GENCMD datei parameterliste

wobei **datei** der Name einer **H86**-Datei ist (Einzel- oder Überlagerungsdatei) und **parameterliste** Schlüsselwörter angibt, die durch Komma oder Leerzeichen getrennt werden. Mögliche Schlüsselwörter sind:

8080 bewirkt die Bildung einer **CMD**-Datei in Form eines **8080**-Speichermodells, bei dem Code- und Datenbereich innerhalb eines Segmentes vermischt sind, unabhängig davon, ob im Quellprogramm **CSEG**- und **DSEG**-Anweisungen verwendet sind.

CODE	Codesegmentzuweisung
DATA	Datensegmentzuweisung
STACK	Stacksegmentzuweisung
EXTRA	Extrasegmentzuweisung
X1, X2, X3, X4	Hilfsbereichzuweisung

Den Schlüsselwörtern, die den jeweiligen Bereich definieren, folgt eine Hexadezimalzahl, die eine Paragraphenadresse oder Segmentlänge in Paragraphen angibt. Diese Werte sind in eckige Klammern einzuschließen und werden durch Komma getrennt. Diesen Werten wird ein einzelner Buchstabe vorangestellt, welcher die Bedeutung des jeweiligen Wortes definiert:

Axxxx Laden auf Absolutadresse xxxx

Bxxxx Bereich beginnt auf xxxx in der **H86-Datei**

Mxxxx Bereich erfordert Speichergröße von minimal xxxx * 16 Byte (xxxx * Paragraphen)

Xxxxx Bereich erfordert Speichergröße von maximal xxxx * 16 Byte (xxxx * Paragraphen)

Hinweis:

– Der A-Wert *muß* für jeden Bereich angegeben werden, der im Speicher fest lokalisiert werden soll. Im *Normalfall* braucht man diesen Wert nicht anzugeben, da SCP 1700 den TPA-Raum intern verwaltet und den Programmen den Speicher zuweist

– Ein B-Wert *muß* angegeben werden, wenn **GENCMD** Dateien verarbeitet, die nicht mit **ASM86** erzeugt sind, z. B. bei Dateien, die keine Informationen zur automatischen Segmentbereichsunterscheidung enthalten.

Beispiele:

B>A:GENCMD SCP 8080 CODE[A40]

Die Datei **SCP.H86** wird in die Datei **SCP.CMD** umgewandelt, wobei das **8080**-Speichermodell zugrunde gelegt wird. Der Codebereich beginnt auf der Paragraphenadresse 40H.

B>A:GENCMD C:TEST CODE[A1000] DATA[M20] EXTRA[B40] STACK[M80] X1[M80]

Eine auf dem Diskettenlaufwerk C befindliche Datei **TEST.H86** wird in die Datei **TEST.CMD** umgewandelt. Der Codebereich beginnt bei der Paragraphenadresse **1000H**, der Datenbereich erfordert einen Speicherbedarf von mindestens **20H** Paragraphen. Der Extrabereich beginnt ab der Paragraphenadresse 40H in der H86-Datei und der Stack- sowie der Zusatzbereich erfordern eine Mindestgröße von **80H** Paragraphen.

6.4.2 LINK86

● **Bedienung**

LINK86 ist ein Dateiverbindungsprogramm, welches verschiedene Objektmodule zu einer **CMD**-Datei verbindet. Hierbei ist es gleichgültig, ob die Objektmodule mit Hilfe des **RASM86** oder anderen Übersetzern (z. B. **FORTRAN 77, C, PASCAL**) erzeugt wurden. Weiterhin erlaubt **LINK86** das Einbinden von Bibliotheksmodulen (**L86**-Dateien), welche indiziert aufgebaut sind. **LINK86** erzeugt drei Dateitypen:

Kommandodatei	(**CMD**-Datei)
Symboltabellendatei	(**SYM**-Datei)
Listdatei	(**MAP**-Datei).

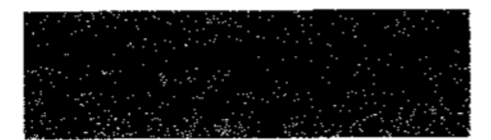
LINK86 wird nach folgendem Bedienschema aufgerufen:

LINK86 {file=}file1{,file2, . . . ,filen}

Wird ein Name vor dem Gleichheitszeichen angegeben, erzeugt **LINK86** die geforderten Ausgabedateien mit ihren dazugehörigen Dateitypen. Wird kein neuer Name angegeben, werden die von **LINK86** erzeugten Dateien mit dem ersten Namen aus der Kommandozeile versehen.



Weiterhin erlaubt **LINK86** das Auslesen öfter wiederkehrender Kommandozeilen aus einer Eingabedatei. Derartige Dateien müssen den Dateityp **INP** haben.



LINK86 kann während seiner Arbeit durch Betätigen einer beliebigen Taste angehalten werden. **LINK86** meldet sich mit folgender Ausschrift:



Die Eingabe von Y veranlaßt **LINK86**, die Arbeit abzubrechen und zum **SCP 1700** zurückzukehren. Ein N setzt die Arbeit von **LINK86** fort.

wird fortgesetzt

Mikroprozessorsystem K 1810 WM 86

Hardware · Software · Applikation (Teil 5)

Prof. Dr. Bernd-Georg Münzer
(wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann,
Wolfgang Kabatzke, Frank Kamrad,
Hellfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock,
Sektion Technische Elektronik,
Wissenschaftsbereich Mikrorechentechnik/
Schaltungstechnik

● **Kommandooptionen**

LINK86 erlaubt eine Reihe von Optionen, die zur Vereinfachung der Arbeit auch als Abkürzungen angegeben werden können. Nachfolgend werden die möglichen Optionen, ihre Abkürzungen (halbfett) und ihre Wirkungen gezeigt.

Option	Wirkung
CODE	steuert Inhalt der CODE-Sektion der CMD-Datei
DATA	steuert Inhalt der DATA-Sektion der CMD-Datei
EXTRA	steuert Inhalt der EXTRA-Sektion der CMD-Datei
STACK	steuert Inhalt der STACK-Sektion der CMD-Datei
X1	steuert Inhalt der X1-Sektion der CMD-Datei
X2	steuert Inhalt der X2-Sektion der CMD-Datei
X3	steuert Inhalt der X3-Sektion der CMD-Datei
X4	steuert Inhalt der X4-Sektion der CMD-Datei
FILL	Einfügen von Nullen und nichtinitialisierten Daten in die CMD-Datei
NOFILL	*kein* Einfügen von Nullen und nichtinitialisierten Daten in die CMD-Datei
INPUT	Lesen einer Kommandozeile aus einer Eingabedatei
MAP	Erzeugen einer MAP-Datei
LIBSYMS	Einfügen von Symbolen aus Bibliotheksdateien in die SYM-Datei
NOLIBSYMS	*kein* Einfügen von Symbolen aus Bibliotheksdateien in die SYM-Datei
LOCALS	Einfügen lokaler Symbole in die SYM-Datei
NOLOCALS	*kein* Einfügen lokaler Symbole in die SYM-Datei
SEARCH	Durchsuchen einer Bibliotheksdatei und Linken der Module, auf die Bezug genommen wird

● **Dateioptionen**

Zur Beeinflussung des Inhaltes der CMD-Datei sind nachfolgend aufgeführte Optionen möglich:

Parameter	Wirkung
ABSOLUTE	absolute Ladeadresse für die Sektion der CMD-Datei
ADDITIONAL	zusätzliche Speicherzuordnung für die Sektion der CMD-Datei
CLASS	Klassen, die in eine Sektion der CMD-Datei eingeschlossen werden
GROUP	Gruppen, die in eine Sektion der CMD-Datei eingeschlossen werden
MAXIMUM	maximale Speicherzuordnung für eine Sektion der CMD-Datei
ORIGIN	Anfang des 1. Segmentes in der CMD-Datei
SEGMENT	Segmente, die in eine Sektion der CMD-Datei eingeschlossen werden

Die MAP-Option erlaubt die Angabe folgender Zusatzoptionen:

OBJMAP/NOOBJMAP

Eingabe/Nichteingabe von Segmentinformationen aus OBJ-Dateien in MAP-Dateien

L86MAP/NOL86MAP

Eingabe/Nichteingabe von Segmentinformationen aus L86-Dateien in MAP-Dateien

ALL

Alle Informationen werden in die MAP-Datei übernommen.

LINK86 hat standardmäßig folgende Optionen eingestellt:

FILL
LOCALS
NOLIBSYMS
OBJMAP
NOL86MAP

● **Optionen der E/A-Geräte**

Die Option ○ legt die Geräte der Ursprungs- und Zieldateien fest. Allgemein wird die Option in folgender Form angegeben:

○td, wobei t den Typ und
d das Gerät angibt.

Als Typ erkennt LINK86 folgende Dateitypen:

C Kommandodatei
L Bibliotheksdatei (L86)
M Listendatei (MAP)
O Objektdatei (OBJ oder L86)
S Symboldatei (SYM)

Als Gerätekennzeichen können die Buchstaben von A bis P für Disketten-/Plattenlaufwerke und die Buchstaben

X für Terminal (CON:)
Y für Drucker (LST:)
Z für Nulldevice (NULL:)

angegeben werden.

Zum Trennen mehrerer ○-Optionen bei der Angabe mehrerer ○-Zeichen müssen Kommas verwendet werden. Wird das ○-Zeichen nur einmal angegeben, sind die ○-Optionen durch Leerzeichen abzugrenzen.

● **Beispiele für die Anwendung von LINK86**

a)

B>A: LINK86 PROG[CODE[SEGMENT[CODE1, CODE2], GROUP[XYU]]]

Wirkung: Einordnung der Segmente CODE1, CODE2 und aller Segmente der Gruppe XYZ in die Code-Sektion der CMD-Datei PROG.CMD

b)

B>A: LINK86 PROG[DATA[ADD[100], MAX[1000]], CODE[ABS[40]]]

Wirkung: Die DATA-Sektion erfordert mindestens 1000H Bytes zusätzlich zu den Daten in der CMD-Datei. Die DATA-Sektion kann bis zu 10000H Bytes des Hauptspeichers verwenden. Die CODE-Sektion muß auf die *Absolutadresse* 400H geladen werden.

c)

B>A: LINK86 TESTX[NOLOCALS], TEST2[LOCALS], TEST3

Wirkung: Erzeugung einer SYM-Datei, die lokale Symbole aus TEST2.OBJ und TEST3.OBJ *aber nicht* aus TEST3.OBJ enthält.

d)

B>A: LINK86 PROG1, PROG2, MATH.L86[S]

Wirkung: Erzeugung von PROG1.CMD durch Verbindung von PROG1.OBJ, PROG2.OBJ und jener Module aus MATH.L86, auf die sich PROG1.OBJ oder PROG2.OBJ beziehen

e)

B>A: LINK86 PROGZ [○SZ, ○OD, ○LB], PROGW
B>A: LINK86 PROGZ [○SZ OD LB], PROGW
B>A: LINK86 PROGZ [○SZODLB], PROGW

Alle drei Kommandozeilen haben die gleiche Wirkung. Sie sind nur in den verschiedenen möglichen Schreibweisen dargestellt.

Wirkung: Erzeugung von PROGZ.CMD auf Laufwerk B:, Unterdrückung von

PROGZ.SYM, Lesen von PROGZ.OBJ und PROGW.OBJ von Laufwerk D: und Suchen der Bibliothek auf Laufwerk B:.

6.5 Die Debugger DDT86 und SID86

DDT86 und SID86 stellen ein aufwärtskompatibles Debugger-Set dar, wobei der DDT86 Bestandteil des SID86 ist. SID86 realisiert gegenüber DDT86 das symbolische Assemblieren und Reassemblieren. Außerdem erlaubt SID86 das Setzen von Protokollpunkten. SID86 nutzt zur symbolischen Testung die jeweilige SYM-Datei. Die Nutzung der SYM-Datei ist optional.

6.5.1 Bedienung von DDT86 und SID86

Der Debugger **DDT86** wird nach folgender Vorschrift aufgerufen und gestartet:

1) **B>A: DDT86**
oder
2) **B>A: DDT86 <file>**.

Das erste Kommando lädt DDT86 und startet es. Nach der Ausgabe des Kommandopromptes (–) ist DDT86 arbeitsbereit. Das zweite Kommando lädt DDT86 und startet es. DDT86 lädt, nachdem es gestartet wurde, die mit <file> spezifizierte Datei. Fehlt der Dateityp, wird .CMD angenommen. Es können keine H86-Dateien geladen werden. Das zweite Kommando kann auch durch folgende Kommandofolge ersetzt werden:

B>A: DDT86
DDT86 Vx.x
–E <file>.

SID86 kann durch eines der folgenden Kommandos gestartet werden:

1) **B>A: SID86**
oder
2) **B>A: SID86 <file>**
oder
3) **B>A: SID86 <file> <symfile>**
oder
4) **B>A: SID86 * <symfile>**.

Die ersten beiden Kommandoformen sind analog denen des DDT86. SID86 meldet sich nach dem Start mit dem Kommandoprompt #. Das dritte Kommando lädt die zu testende Datei und die Symboldatei. Das vierte Kommando lädt nur die Symboldatei. Die Kommandofolgen 2.), 3.) und 4.) lassen sich durch folgende ersetzen:

2) **B>A: SID86**
SID86 Vx.x
#E <file>
3) **B>A: SID86**
SID86 Vx.x
#E <file> <symfile>
4) **B>A: SID86**
SID86 Vx.x
#E * <symfile>.

Abkürzungen zu den Kommandos

s	20-Bit-Anfangsadresse
d	20-Bit-Zieladresse
f	16-Bit-Offset im spezifizierten Segment
b	Bytewert
w	Wortwert
bn	Unterbrechungspunkt
W	Anzeige Wortweise
S	Anzeige Segmentregister
R	Registerspezifikation
F	Flagspezifikation

6.5.2 Kommandos von DDT86 und SID86

Folgende Kommandos sind möglich:

A *Assembling*
As
Eingabe von Assembleranweisungen, s = 20-Bit-Adresse, wo die Assemblierung beginnt. Es gilt im wesentlichen die Assemblernotation nach ASM86/RASM86. DDT86 kann für s *nur* absolute hexadezimale Werte verarbeiten, während SID86 auch symbolische Ausdrücke verarbeiten kann.

Beispiele:
DDT86
–A1000:0
1000:0 MOV DX, 100
1000:3.
–
SID86
#A1000:0
1000:0 MOV DX, 100
1000:3
#
oder, falls 100H = WERT
#A1000:0
1000:0 MOV DX,.WERT
1000:3.
#

B *Blockcompare*
Bs1, f1, s2
s1 = 20-Bit-Adresse des Beginns des ersten Speicherblockes
f1 = Offset des letzten Bytes des Speicherblockes
s2 = 20-Bit-Adresse des Beginnes des 2. Speicherbereiches
Jede Differenz zwischen den Speicherbereichen wird auf dem Bildschirm angezeigt.

Beispiel:
B1000:0, 2FF, 2000:0
Vergleich von 300H Bytes ab 1000:0H mit dem Block ab 2000:0H.

D *Display*
a) **D**
b) **Ds**
c) **Ds,f**
d) **DW**
e) **DWs**
f) **DWs,f**

E *Programm laden, Symbole laden*
a) **E <file>**
b) **E <file> <symfile>** nur SID86
c) **E * <symfile>** nur SID86
d) **E**

Die Form a) lädt die durch <file> angegebene Datei. Wenn die Datei vollständig geladen ist, zeigen DDT86/SID86 die Start- und Endadresse jedes geladenen Segmentes an. Die Formen b) und c) sind bereits in 6.5.1 erläutert. Die Form e) gibt alle Segmentbereiche vorher geladener Programme wieder frei.

F *Fill*
a) **Fs, f, b**
b) **FWs, f, w**

Die Form a) speichert den 8-Bit-Wert b von s bis f. In der Form b) wird der 16-Bit-Wert w von s bis f gespeichert (in der Standardform: Low-Teil, High-Teil).

Beispiel:
F1000:0, 2FF, 55
füllt den Speicherbereich von 1000:0H bis 1000:2FFH mit 55H. Wird die Segmentadresse weggelassen, wird das aktuelle Segment angenommen.

G *Go (Programmstart)*
a) **G**
b) **G, b1**
c) **G, b1, b2**
d) **Gs**
e) **Gs, b1**
f) **Gs, b1, b2**
g) **–G** (nur SID86)

Die Formen a), b) und c) sind ohne spezifizierten Startpunkt. Ihr Startpunkt wird aus dem aktuellen CS und IP gebildet. Bei den Formen d) bis f) wird eine Startadresse mit angegeben. Die Formen b), c), e) und f) geben einen oder mehrere Unterbrechungspunkte an. Die Form g) unterdrückt die Ausgabe von Protokollpunkten (nur SID86).

L *List*
a) **L**
b) **Ls**
c) **Ls, f**

Ohne Parameter gibt das L-Kommando 12 Zeilen reassemblierten Maschinencode ab der aktuellen Adresse aus. Ist der Parameter s gesetzt, wird vor der Ausgabe die Anfangsadresse auf s gesetzt, und es werden 12 Zeilen ausgegeben. Die letzte Form reassembliert Maschinencode von s bis f und gibt ihn fortlaufend aus.

M *Move*
Ms, f, d
Das M-Kommando bewirkt den Transport eines Speicherbereiches von s bis f nach d. Falls für d kein Segment spezifiziert wurde, wird der gleiche Wert wie bei s angenommen.

R *Read*
R <file>
Das R-Kommando liest eine Datei in einen zusammenhängenden Speicherbereich (ohne Bereichsaufspaltung). Das R-Kommando gibt keine Speicherbereiche frei, die durch frühere R- oder E-Kommandos belegt wurden. Die Anzahl der zu ladenden Dateien ist auf 7 begrenzt.

S *Substitute*
a) **Ss**
b) **Sw**

Mit Hilfe des S-Kommandos kann der Inhalt von Bytes oder Worten im Speicher geändert werden. Die Speicheradressen und die alten Speicherinhalte werden nach Kommandoeingabe angezeigt. Eine Eingabe gültiger Hexeadezimalwerte überschreibt diese. Die Eingabe von nur RETURN läßt den Inhalt unverändert. Eine Anzeige bzw. Änderung ist bis zur Eingabe eines Punktes (.) oder eines unerlaubten Wertes möglich.

T *Trace*
a) **T**
b) **Tn**
c) **TS**
d) **TSn**

Das T-Kommando bewirkt eine Programmverfolgung im Tracemodus (für n = 1 – 0FFFFH) mit Angabe der Inhalte der CPU-Register. Bei den Formen a) und b) werden die Segmentregister nicht mit ausgegeben. Wird der Wert n nicht angegeben, wird nur ein Befehl ausgeführt. Zusätzlich wird der jeweils nächste Befehl in reassemblierter Form mit ausgegeben.

U *Untrace*
a) **U**
b) **Un**

c) **Us**
d) **USn**
Das U-Kommando ist mit dem T-Kommando identisch mit der Ausnahme, daß die Inhalte der CPU-Register *nur* vor der Ausführung des *ersten* Befehls angezeigt werden.
V *Value*
Das V-Kommando zeigt die aktuell durch das jeweilige Programm belegten Segmente an.
W *Write*
a) **W <file>**
b) **W <file>s, f**
Das W-Kommando schreibt den Inhalt eines zusammenhängenden Speicherbereiches zum Massenspeicher. Werden s und f nicht angegeben, übernehmen SID86/DDT86 die Werte von der letzten mit einem R-Kommando gelesenen Datei. *Ist die mit einem W-Kommando zu schreibende Datei bereits vorhanden, wird sie überschrieben!*
X *Anzeige der CPU-Register*
a) **X**
b) **XR**
c) **XF**
Das X-Kommando erlaubt die Anzeige des CPU-Statusses. Durch Spezifikation können einzelne Register (Form b)) oder Flags (Form c)) angezeigt oder geändert werden.
P *Pass-Point* (nur SID86)
a) **Pd, n**
b) **Pd**
c) **–Pd**
d) **–P**
e) **P**
Das P-Kommando setzt, löscht und zeigt Protokollierpunkte an. Die Formen a) und b) werden zum Setzen von Protokollierpunkten genutzt. Der Wert n in der Form a) gibt einen Durchlaufzählerwert an. Die Formen c) und d) werden genutzt, um Protokollierpunkte zu löschen. Die Form d) löscht alle Protokollierpunkte. Die Form e) zeigt alle aktiven Protokollierpunkte an.

7. Koprozessoren

Eine der wichtigsten Methoden zur Erhöhung der Effektivität von Mikrorechnersystemen ist die Parallelarbeit von mehreren Prozessoren in einem Rechner bei wechselseitigem Austausch von Informationen.
Informationsaustausch kann über den gemeinsamen Speicher oder E/A-Ports stattfinden. Das erste Architekturprinzip bezeichnet man als eng gekoppeltes System (tightly/closely/coupled microprocessor systems) und das zweite Prinzip als lose gekoppeltes System (loosely coupled microprocessor systems).
Die 8086-CPU besitzt einen 6-Byte-FIFO-Instruction-Queue in der Bus-Interface-Unit, welcher dem Prozessor ein vorausschauendes Befehlsholen ermöglicht. Solcher Warteschlangenmechanismus der CPU trägt in Multiprozessorsystemen zur Erhöhung des Parallelitätsgrades bei.
In diesem Beitrag sollen die spezialisierten Koprozessoren des 8086-Systems
– Arithmetikkoprozessor 8087
– Ein-/Ausgabe-Prozessor 8089
vorgestellt werden, die mit der 8086-CPU ein eng gekoppeltes Multiprozessorsystem darstellen.

Tafel 7.1 Zusammenstellung von Befehlsausführungszeiten

Gleitkomma-operation	Ausführungszeit in µs	
	8087 (5MHz)	8086 Emulator
ADD/SUBTRACT	14/18	1600
Multiply (single precision)	19	1600
Multiply (extended precision)	27	2100
Divide	39	3200
Compare	9	1300
Load (double precision)	10	1700
Store (double precision)	21	1200
Square root	36	19600
Tangent	90	13000
Exponentiation	100	17100

7.1 Systemkonfiguration mit Arithmetikkoprozessor 8087

Der Arithmetikkoprozessor 8087 ist nur in Zusammenarbeit mit der 8086/88-CPU einsetzbar. Seine interne Struktur ermöglicht die Ausführung von numerischen Operationen mit hoher Geschwindigkeit und Präzision. Tafel 7.1 zeigt eine Zusammenstellung von Befehlsausführungszeiten beim Koprozessor 8087 im Vergleich zur 8086-CPU-Emulation. Für den Anwender erscheint die Verbindung der CPU mit dem Arithmetikkoprozessor als ein komplexer Mikroprozessor mit vergrößertem Befehlsvorrat. Der 8087 liefert dem System neue Datentypen, neue Register und 68 neue Befehle. Er arbeitet parallel zur Master-CPU, das heißt, er dekodiert parallel den Befehlsstrom, führt aber nur diejenigen Befehle aus, die für ihn bestimmt sind (ESCAPE-Befehle). Die schaltungstechnische Realisierung der Verbindung von Arithmetikkoprozessor 8087 mit der 8086-CPU zeigt Bild 7.1. Die Statussignale $\overline{S0}$–$\overline{S2}$ und die Queue-Statussignale QS0–QS1 ermöglichen dem 8087 das „Mithören" und Dekodieren der Befehle parallel zur CPU.
Zum Synchronisieren wird das BUSY-Signal benutzt, das mit dem $\overline{TEST}$-Eingang der CPU verbunden ist und von der CPU abgefragt wird.
Der Koprozessor kann in einem ERROR- oder EXCEPTION-Fall die Programmbearbeitung der CPU mit einem Interrupt unterbrechen, der über den programmierbaren Interrupt-Controller 8259A an die CPU weitergeleitet wird.
Wie aus Bild 7.1 ersichtlich, werden beide Prozessoren ohne zusätzlichen Hardwareaufwand miteinander verbunden. Der Adreß-, Daten- und Steuerbus wird von beiden Prozessoren gemeinsam genutzt. Diese Konfiguration erfordert eine Arbitrierungslogik zur Kontrolle der aktuellen Buszuweisung.
Der Busanforderungs-, Busübernahme- und Busrückgabezyklus wird mit Hilfe der bidirektionalen $\overline{RQ}/\overline{GT}$-Leitung realisiert (Bild 7.2). Dieses Signal muß an $\overline{RQ}/\overline{GT0}$ oder $\overline{RQ}/\overline{GT1}$ der CPU angeschlossen werden. Der Request/Grant-Sequenz läuft in drei Etappen ab:
- Senden des ein Taktzyklus langen Request-Impulses an die CPU; Bedeutung: Der 8087 oder ein anderer Master fordert den lokalen Bus an.
- Der 8087 wartet auf den Grant-Impuls von der CPU; der 8087 startet einen Buszyklus nach Eintreffen des Grant-Impulses und übernimmt damit die Buskontrolle.
  Falls der Request-Impuls von einem anderen Master (8089) kam, leitet er den Impuls an seinen $\overline{RQ}/\overline{GT1}$-Anschluß weiter (Bild 7.1).
- der 8087 sendet den Release-Impuls an die CPU oder schaltet den von einem anderen Master auf der Linie $\overline{RQ}/\overline{GT1}$ empfangenen Release-Impuls durch, die CPU übernimmt wieder die Buskontrolle.

7.2 Busstruktur des Arithmetikkoprozessors 8087

Die Busstruktur des 8087 ist identisch mit der der 8086-CPU im Maximum-Mode (ver-

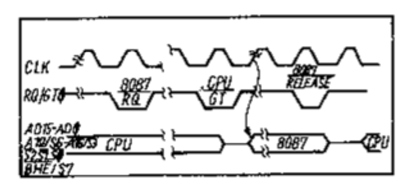


Bild 7.2 RQ/GT-Zeitdiagramm

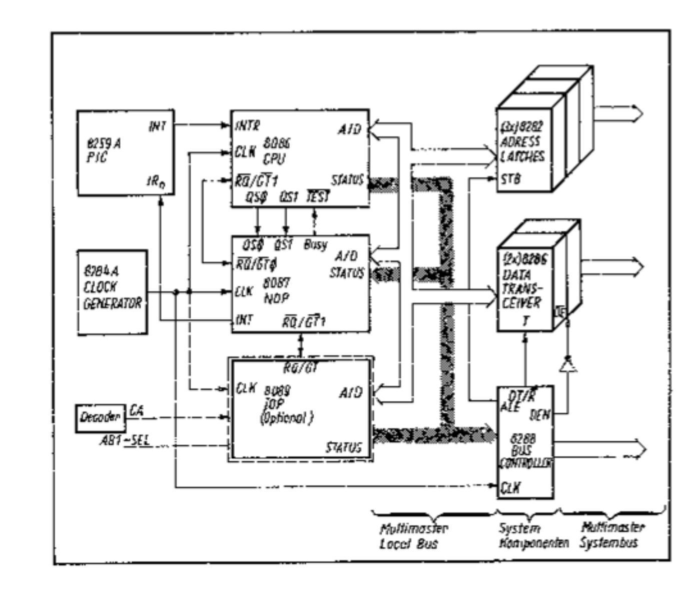


Bild 7.1 Systemkonfiguration mit Arithmetikkoprozessor 8087 und mit I/O-Prozessor 8089

gleiche Abschnitt 1.2). Der Koprozessor nutzt gemeinsam mit der CPU die Systemkomponenten: Buscontroller, Clockengenerator, Octal-Latches usw. Die Statussignale $\overline{S0}$–$\overline{S2}$ werden vom 8087 wie folgt dekodiert:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bedeutung
0	X	X	nicht genutzt
1	0	0	nicht genutzt
1	0	1	Speicherlesen
1	1	0	Speicherschreiben
1	1	1	passiv

Wenn der 8087 die Buskontrolle besitzt, sind die Statussignale S6, S4 und S3 = High, während S5 = Low ist (siehe dazu Abschnitt 2.1.2). Im passiven Zustand wird vom 8087 das Statussignal S6 abgefragt, welches die Information enthält, ob die Buskontrolle von der CPU oder einem Koprozessor durchgeführt wird. S7 wird mit $\overline{BHE}$ gemultiplext und besitzt für alle 8087-Buszyklen den Wert von $\overline{BHE}$. Der 8087 enthält einen Instruction-Queue, der mit dem Queue der 8086-CPU identisch ist. Der 8087 kontrolliert die Statussignale QS0 und QS1, wodurch die Abarbeitung der sich in der Warteschlange befindenden Befehle synchron mit der 8086-CPU ablaufen kann. Diese Signale werden wie folgt dekodiert:

QS1	QS0	Bedeutung
0	0	keine Operation
0	1	erstes Byte vom Operationscode vom Queue entnommen
1	0	Queue leer
1	1	nachfolgendes Byte vom Queue entnommen

7.3 Interne Architektur des 8087

In der internen Struktur des Arithmetikkoprozessors unterscheidet man zwei Module:

- Steuereinheit (Control Unit; CU)
- Recheneinheit (Numeric Execution Unit; NEU)

Die Struktur des Arithmetikkoprozessors ist im Bild 7.3 dargestellt.

Control Unit

Die CU holt und klassifiziert die Befehle, liest bzw. schreibt die Operanden und nimmt die Synchronisation mit der CPU vor. Die 8087-Befehle befinden sich im Befehlsstrom des CPU-Programms. Die CU des Koprozessors erhält durch den Status $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ und QS0, QS1 die Information, wann ein Befehlsholezyklus (Fetch) stattfindet. Die im Befehlsholezyklus gelesenen Daten (Operationscode) werden von der CU gelesen und dekodiert. Die fünft höchsten Bits aller 8087-Operationscodes sind identisch (ESCAPE-Präfix 11011 ...) und kennzeichnen damit den Befehl als ESCAPE-Befehl. Wenn ein ESCAPE-Befehl erkannt wird, wird er von der CU dekodiert und von der CU bzw. von der NEU ausgeführt. Die Master-CPU analysiert auch alle ESCAPE-Befehle, und im Falle eines Operandentransfers vom oder zum Speicher rechnet sie die entsprechende Operandenadresse aus. Die berechnete Adresse wird in einem „leeren Lesezyklus" („dummy read") auf den Adreßbus ausgegeben und vom 8087 übernommen, wobei zwei Fälle unterschieden werden:

– *Operanden sollen gelesen werden*

Das im „leeren Lesezyklus" gelesene Datenwort wird vom 8087 übernommen. Falls der Operand länger als ein Wort ist, übernimmt der 8087 die Buskontrolle, und auf Grundlage der übernommenen Operandenadresse führt er weitere Lesezyklen durch.

– *Operanden sollen geschrieben werden*

Das im „leeren Lesezyklus" auf den Datenbus gelegte Datenwort wird ignoriert. Wenn der 8087 schreiben will, übernimmt er die Buskontrolle und führt Schreibzyklen durch, die bei der übernommenen Adresse beginnen.

Numeric Execution Unit NEU

Die NEU führt alle Befehle aus, die sich auf die Stack-Register beziehen. Dazu gehören arithmetische, logische, logarithmische, trigonometrische, Konstanten- sowie Datentransferbefehle. Der interne Datenbus der NEU ist 84 Bit breit (68 Bit Mantisse, 15 Bit Exponent, 1 Bit Vorzeichen). Während einer Befehlsabarbeitung wird das BUSY-Signal high-aktiv geschaltet und kann von der CPU abgefragt werden.

Registersatz

Der Registersatz des Arithmetikkoprozessors beinhaltet acht 80 Bit breite Stack-Register, die in folgende Felder aufgeteilt werden:

64 Bit Mantisse
15 Bit Exponent
1 Bit Vorzeichen

die vom Format dem „temporary real data type" entsprechen.

Status Word

Das Statuswort ist 16 Bit breit und beinhaltet die Informationen über den Zustand des Koprozessors. Der Status kann getestet, im Speicher mit dem Befehl FSTSW abgespeichert und vom CPU-Programm kontrolliert werden. Das Statuswort und die Bedeutung der einzelnen Bits ist in den Tafeln 7.2 und 7.3 dargestellt.

Control Word

Das Steuerwort ist 16 Bit breit, wird mit dem Befehl FLDCW geladen und dient zum Initialisieren des Koprozessors. Die Bedeutung einzelner Bits ist in Tafel 7.4 dargestellt.

Das niederwertige Byte des Steuerwortes konfiguriert Interrupt- und Exception-Verhalten des 8087. Mit Bit 7 werden generell die Interrupts gesperrt (High) oder freigegeben (Low). Mit den Bits 0–5 können die Exceptionbedingungen maskiert werden. Jede Exzeptionbedingung kann Ursache eines Interrupts sein, falls Interrupt freigegeben und Exception nicht maskiert (Low) wurde. Das Steuerwort kann mit dem Befehl FSTCW in den Speicher geladen werden.

Tag Word

Das Tag Word kennzeichnet den Inhalt der Register (Tafel 7.5) und kann mit den Befehlen FSTSW, FSAVE und FSTENV im Systemspeicher abgelegt werden.

Exception Pointers

Die Exception Pointers (Tafel 7.6) sind für die vom Programmierer geschriebenen Error-Behandlungsprogramme von Bedeutung. Während die NEU einen Befehl ausführt, speichert die CU die Operandenadresse

Bild 7.3 8087-Blockdiagramm

Tafel 7.2 8087 Status Word

Bit	Symbol	Bedeutung
0	IE	Invalid Operation
1	DE	Denormalized Operand
2	ZE	Zero Divide
3	OE	Overflow
4	UE	Underflow
5	PE	Precision
6	XXX	(reserviert)
7	IR	Interrupt Request
8	C0	siehe Tafel 7.3
9	C1	siehe Tafel 7.3
10	C2	siehe Tafel 7.3
11 12 13	TOP	Top of Stack Pointer 0 0 0 Register 0 TOP 0 0 1 Register 1 TOP 0 1 0 Register 2 TOP 0 1 1 Register 3 TOP 1 0 0 Register 4 TOP 1 0 1 Register 5 TOP 1 1 0 Register 6 TOP 1 1 1 Register 7 TOP
14	C3	siehe Tafel 7.3
15	B	NEU BUSY

Tafel 7.3 Interpretation der Condition Codes

[illegible]	[illegible]	[illegible]
[illegible]	0 X X 0	A > B
	0 X X 1	A < B
	1 X X 0	A = B
	1 X X 1	A ? B (nicht vergleichbar)
[illegible]	U 0 U U	Complete reduction
	U 1 U U	Incomplete reduction
[illegible]	0 0 0 0	Valid, positiv, unnormalized
	0 0 0 1	Invalid, positiv, exponent – 0
	0 0 1 0	Valid, negativ, exponent = 0
	0 0 1 1	Invalid, negative, exponent = 0
	0 1 0 0	Valid, positiv, normalized
	0 1 0 1	Infinity, positiv
	0 1 1 0	Valid, negativ, normalized
	0 1 1 1	Infinity, negative
	1 0 0 0	Zero, positive
	1 0 0 1	Empty
	1 0 1 0	Zero, negative
	1 0 1 1	Empty
	1 1 0 0	Invalid, positiv, exponent – 0
	1 1 0 1	Empty
	1 1 1 0	Invalid, negativ, exponent = 0
	1 1 1 1	Empty

U – Wert nach der Operation undefiniert
X – Wert wird durch die Operation nicht verändert

Tafel 7.4 Control Word

Bit	Symbol	Bedeutung	
0	IM	Invalid Operation	
1	DM	Denormalized Operand	
2	ZM	Zerodivide	Exception
3	OM	Overflow	Masks
4	UM	Underflow	1 – Exception
5	PM	Precision	is masked
6	X	(Reserved)	
7	IEM	Interrupt Enable Mask	0 Int enabled 1 Int masked
8, 9	PC	Precision Control	0 0 24 Bit 0 1 (Reserved) 0 1 53 Bit 1 1 64 Bit
10, 11	RC	Rounding Control	0 0 To Nearest or Even 0 1 Round down (toward ∞) 1 0 Round up (toward + ∞) 1 1 Chop (truncate toward zero)
12	IC	Infinity Control	0 Projective 1 Affine
13, 14, 15		Reserved	

(falls vorhanden) und den Befehlscode in den Exception Pointers ab. Diese Daten können dann im Speicher abgelegt werden.

7.4 Datentypen und -formate

Der Arithmetikkoprozessor 8087 arbeitet mit sieben Datenformaten, die in drei Klassen eingeteilt werden:

– binary integers
– packed decimal integers
– binary reals

Bild 7.4 zeigt den Aufbau der 8087-Datenformate.

Im Bild 7.5 ist dargestellt, wie die einzelnen Datenformate im Speicher des 8086-Systems abgelegt werden.

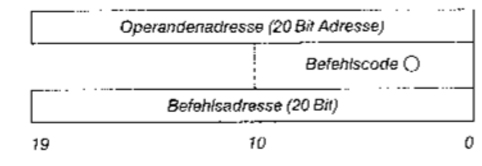
Tafel 7.5 Tag Word

15 ... 0

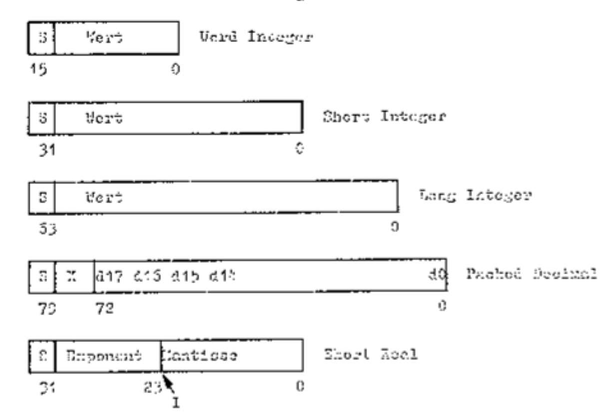
TAG(7)	TAG(6)	TAG(5)	TAG(4)	TAG(3)	TAG(2)	TAG(1)	TAG(0)

TAG-Wert: 0 0 Valid (normal or unnormal)
0 1 Zero (True)
1 0 Special (Not-a-Number, unnormal)
1 1 Empty

Tafel 7.6 Exception Pointers Format



○ Es werden nur 11 Bits des Befehlscodes gespeichert, da die 5 höchstwertigen immer den Wert 11011B haben.



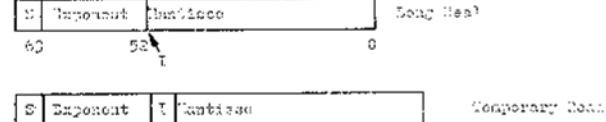
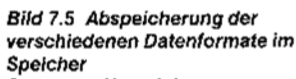
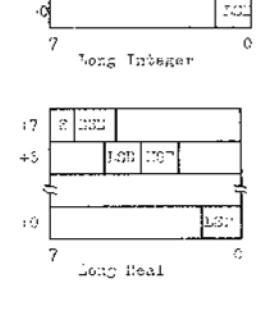


Bild 7.4 Datenformate des 8087
S – Vorzeichen (0 = positiv, 1 = negativ)
dn – Dezimalzahlen (zwei pro Byte)
X – Bits haben keine Bedeutung; werden ignoriert beim Laden; beim Abspeichern gleich Null
I – Integer-Bit der Mantisse – Komma





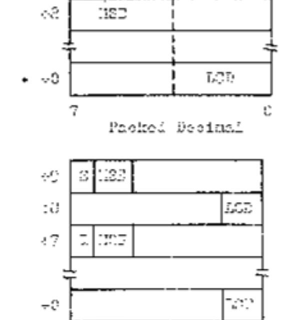


Bild 7.5 Abspeicherung der verschiedenen Datenformate im Speicher
S – Vorzeichen;
MSB/LSB – Most/least significant bit;
MSD/LSD – Most/least significant decimal digit;
MSE/LSE – Most/least significant exponent bit;
MSF/LSF – Most/least significant fraction;
I – Integer bit

Beim Laden der Daten in die internen Stack-Register und beim Abspeichern der Daten wird eine automatische Umformatierung vorgenommen.

Beispiel 1
Der Befehl FILD konvertiert die im Systemspeicher im Integer-Format abgespeicherten Daten in das 8087-interne Temporary-real-Format und lädt sie in die Stack-Register.

Beispiel 2
Der Befehl FBSTP konvertiert den Inhalt des „Top of Stack" in das Packed-decimal-Format und speichert den Wert im Systemspeicher ab.

7.5 Assemblerbefehle des 8087

Die 8087-Befehle führen Operationen im Speicher oder in internen Stack-Registern aus. Es werden folgende Gruppen von Befehlen unterschieden:

- Transferbefehle: load, store, exchange
- Arithmetikbefehle: add, subtract, multiply, divide, square root, scale usw.
- Vergleichbefehle: test, examine, compare
- Funktionen: tang, arctang, $2^x - 1$, $Y * \log_2 (X + 1)$, $Y * \log_2 (x)$
- Konstanten: 0, 1, π, $\log_{10} 2$, $\log_e 2$, $\log_2 10$, $\log_2 e$
- Prozessorsteuerbefehle: FINIT, Load Control Word, Store Control Word, Enable/Disable Interrupt, Clear Exception.

7.5.1 Datentransportbefehle

Mit Transportbefehlen können die internen 10-Byte-Daten innerhalb des Registerstapels auf die Stapelspitze umgespeichert oder bei gleichzeitiger Konvertierung in das Anwenderdatenformat vom Speicher gelesen oder in den Speicher geschrieben werden. Der Austausch mit dem Speicher erfolgt nur über die Stapelspitze. Für Speicheroperanden wird der Datentyp (Integer, Real oder Dezimal) in der mnemonischen Befehlsbeschreibung und die Datenlänge in der Operandenbeschreibung in Erweiterung der 8086-Assemblernotierung (z. B. DWORD PTR [BX]) angegeben. Der Befehl FLD führt eine Ladeoperation für Realdaten auf die Stapelspitze aus. Dabei wird der Stapelzeiger vor dem Laden auf die nächst kleinere Registernummer gestellt, so daß vorangegangene Stapeleintragungen erhalten bleiben. Ladeoperationen mit Quelloperanden im Registerstapel kopieren diese auf die Stapelspitze. Ladeoperationen aus dem Speicher sind für das 10-, 8- und 4-Byte-Format möglich. Bei 4- und 8-Byte-Realdaten erfolgt die Konvertierung auf das interne 10-Byte-Format.

Beispiele:

FLD ST(3) ; ST(0) = ST(3)
FLD ST(0) ; Duplizieren der Stapelspitze
FLD DWORD PTR [BX]; 4-Byte-Real-Format
FLD QWORD PTR [SI] ; 8-Byte-Real-Format
FLD TBYTE PTR [DI] ; 10-Byte-Real-Format

Für die Eingabe von Integer- und Dezimaldaten auf die Stapelspitze existieren Befehle mit der mnemonischen Beschreibung FILD und FBLD, die ebenfalls die Konvertierung in das interne 10-Byte-Format einschließen.

Beispiele:

FILD WORD PTR [BP] ; 2-Byte-Integer-Format
FILD DWORD PTR [SI]; 4-Byte-Integer-Format
FILD QWORD PTR [DI]; 8-Byte-Integer-Format
FBLD TBYTE PTR [BX] ; 10-Byte-Integer-Format

Die Operation FST überträgt Realdaten von der Stapelspitze in andere Register oder in den Speicher, wobei der Quelloperand auf der Stapelspitze erhalten bleibt. Die Übertragung in den Speicher ist nur für 4- und 8-Byte-Daten zugelassen. Die Verkürzung des 10-Byte-Formates unterliegt einer von vier Rundungsvorschriften, die mit dem 8087-Steuerwort eingestellt wird.

Beispiele:

FST ST(4) ; ST(4) = ST(0)
FST DWORD PTR [DI]; 4-Byte-Real-Format

Die Abspeicherung von Integer-Daten mit dem Befehl FIST ist für 2- und 4-Byte-Ergebnisdarstellungen möglich.

Beispiele:

FIST WORD PTR [SI] ; 2-Byte-Integer
FIST DWORD PTR [DI]; 4-Byte-Integer

Von größerer praktischer Bedeutung ist die Abspeicherung der Stapelspitze in Verbindung mit einer anschließenden POP-Operation, die den Registerstapelzeiger zurücksetzt. Der entsprechende Befehl FSTP für Realdaten schließt auch die Abspeicherung von 10-Byte-Daten im Speicher ein.

Beispiele:

FSTP ST(0) ; POP-Operation ohne Datentransport
FSTP TBYTE PTR [BX]; 10-Byte-Real-Daten abspeichern und POP-Operation

Die Befehle FISTP und FBSTP sind für alle Integer-Formate und das 10-Byte-Dezimal-Format gültig.

Beispiele:

FISTP QWORD PTR [SI]; 8-Byte-Integer- und ; 10-Byte-Daten
FBSTP TBYTE PTR [DI] ; abspeichern ; und POP-Operation

Der Befehl FXCH erlaubt den Datenaustausch zwischen der Stapelspitze und Stapelregistern oder Operanden im Speicher.

7.5.2 Arithmetische Grundoperationen

Die arithmetischen Operationen des 8087 enthalten die vier Grundoperationen und zwei reverse Formen für die Subtraktion und Division, bei denen die Operanden vor der Operation vertauscht werden.
In Abhängigkeit von der Form der Operandenvorgabe können die folgenden Befehlsformen unterschieden werden:

□ Stapelabarbeitung

Die Operation bezieht sich auf die letzten beiden Stapeleintragungen ST(0) und ST(1). Das Ergebnis steht nach dem Erhöhen des Stapelzeigers um 1 an der Stapelspitze (Bild 7.6a). Die Operation wird im Assemblerprogramm ohne Operandenangabe angegeben:

FADD ; Addition im Stapel
FSUB ; Subtraktion im Stapel ; ST(1) – ST0
FMUL ; Multiplikation im Stapel
FDIV ; Division im Stapel ST(1)/ST
FSUBR; reverse Subtraktion im Stapel ; ST0 · ST(1)
FDIVR; reverse Division im Stapel ; ST0/St(1)

Bild 7.6
a) FADD
b) FADD ST(3), ST
c) FADDP ST(2), ST

□ Registerabarbeitung

Die Operanden stehen im Register ST(0) und in einem beliebigen anderen Register. Das Ergebnis kann auf einem der Vorgaberegister abgelegt werden (Bild 7.6b).

FADD ST, ST(i)	oder	**FADD** ST(i), ST
FSUB ST, ST(i)	oder	**FSUB** ST(i), ST
FMUL ST, ST(i)	oder	**FMUL** ST(i), ST
FDIV ST, ST(i)	oder	**FDIV** ST(i), ST
FSUBR ST, ST(i)	oder	**FSUBR** ST(i), ST
FDIVR ST, ST(i)	oder	**FDIVR** ST(i), ST

□ Registerabarbeitung mit Stapel-POP-Operation

Wenn der im Register ST(0) vorgegebene Operand nur für diese Operation benötigt wird, kann mit einer POP-Operation nach der Berechnung der in ST(0) vorgegebene Operand aus dem Registerstapel herausgeschoben werden (Bild 7.6c). Bei der Angabe des Registers ST(1) als Ergebnisregister steht das Ergebnis nach der Operation in ST(0), da für alle Register der relative Registerzeiger dekrementiert wird.

FADDP ST(i), ST ; Addition im Stapel mit POP
FSUBP ST(i), ST ; Subtraktion im Stapel mit POP
FMULP ST(i), ST ; Multiplikation im Stapel mit POP
FDIVP ST(i), ST ; Division im Stapel mit POP
FSUBRP ST(i), ST; reverse Subtraktion im Stapel mit POP
FDIVRP ST(i), ST ; reverse Division im Stapel mit POP

□ Operation mit Speicheroperand

Die Operation bezieht sich auf die Stapelspitze und einen Operanden im Speicher im 4- oder 8-Byte-Real- oder im 2- oder 4-Byte-Integer-Format. Für die Integerformate gel-

ten die Bezeichnungen FIADD, FISUB, FIMUL, FIDIV, FISUBR und FIDIVR.

Beispiele:
FIADD WORD PTR [SI] ; **Addition mit 2-Byte -Integer-Daten**
FDIV DWORD PTR [BX]; Division mit 4-Byte -Real-Daten

7.5.3 Spezielle arithmetische Operationen
Die Berechnung der Quadratwurzel mit dem Befehl FSQRT überschreibt das Argument an der Stapelspitze. Die Vorgabe von Argumenten < 0 führt zur Fehlermarkierung. Die Multiplikation und Division mit ganzzahligen Potenzen von 2 kann durch Addition auf den Exponenten der internen Gleitpunktdualdarstellung einfach und schnell ausgeführt werden. Der entsprechende Befehl FSCALE benutzt den Inhalt der vorletzten Stapeleintragung ST(1) als vorzeichenbehafteten 2-Byte-Integer-Skalierungsfaktor für ST(0). Die Operation FPREM ergibt eine Modulo-Division von ST(0) mit dem Modulus in ST(1). Die Operation wird durch sukzessive bewertete Subtraktionen ausgeführt, bis der verbleibende Rest kleiner als der Modulus ist. Das Vorzeichen des Ergebnisses stimmt mit dem der Vorgabe in ST überein.
Die Rundung auf Integerwerte mit dem Befehl FRNDINT unterliegt genau wie die Konvertierung der internen Gleitpunktdarstellung in die Integerdarstellungen der mit dem 8087-Steuerwort eingestellten Rundungsvorschrift.
Der Befehl FXTRACT zerlegt den Wert an der Stapelspitze in den Wert des originalen (nicht verschobenen) Dualexponenten und den Mantissenwert. Der Mantissenwert an der neuen Stapelspitze ST(0) und der Wert des vorzeichenbehafteten absoluten Exponenten in ST(1) sind in dem internen 10-Byte-Dualdatenformat (mit Exponentenverschiebung) dargestellt.
Die Befehle FABS und FCHS bilden den Betrag und das Komplement von ST(0) durch Beeinflussung des Vorzeichenbits.

7.5.4 Vergleichsbefehle
8087-Berechnungsergebnisse können nach der Auslagerung in den Speicher mit 8086-Vergleichsbefehlen überprüft werden und Programmverzweigungen steuern.
Rechenzeitgünstiger sind Vergleichsoperationen im 8087-Registerstapel. Die Vergleichsergebnisse werden im Bedingungscodefeld des 8087-Statuswortes markiert. Nach der Abspeicherung des Statuswortes wird der Bedingungscode mit 8086-Befehlen ausgewertet.
Die Vergleichsoperation FCOM bezieht sich auf die bei den arithmetischen Grundoperationen beschriebenen Operandenformen der Stapel- und Registerabarbeitung und der 4- und 8-Byte-Realdaten im Speicher.

Beispiele:
FCOM ; Vergleich ST mit ST(1)
FCOM ST(5) ; Vergleich ST mit ST(5)
FCOM DWORD PTR [SI]; Vergleich ST mit 4-Byte-Real-Daten

Für den Vergleich mit 2- oder 4-Byte-Integer-Daten im Speicher existiert der Befehl FICOM.
Vergleichsoperationen mit anschließender POP-Operation FCOMP und FICOMP löschen den Operanden an der Stapelspitze. Für den Vergleich von ST mit ST(1) existiert zusätzlich der Befehl FCOMPP mit der anschließenden zweimaligen POP-Operation. Den Vergleich von ST mit dem Wert 0 vollzieht der Befehl FTST. Das Vergleichsergebnis für alle Vergleichsoperationen enthält das Bedingungscodefeld des Statuswortes:

C3	**C0**	
0	0	ST > 2. Operand
0	1	ST < 2. Operand
1	0	ST = 2. Operand
1	1	kein Vergleich möglich

Eine umfangreichere Statusinformation von ST ergibt der Befehl FXAM in den Bits C0, C1, C2 und C3 des Statuswortes.

7.5.5 Transzendente Funktionen
Die trigonometrischen und zyklometrischen Standardfunktionen lassen sich auf die Tangens- und Arcustangens-Funktion zurückführen. Die Berechnungsgrundlage der Funktion tan(Z) liefert der Befehl FPTAN mit einer Argumentenvorgabe $0 < Z < \frac{\pi}{4}$ in der Stapelspitze. Das Ergebnis entsteht in Form zweier Werte X (in ST) und Y (in ST(1)). Der Quotient Y/X ergibt den Tangenswert. Die Umkehrfunktion FPATAN berechnet Z = arctan (Y/X) = arctan (ST(1)/ST) mit der Bedingung $0 < Y < X < \infty$. Das Ergebnis überschreibt beide Vorgabeoperanden.
Die Exponentialfunktion zur Basis 2 kann als Grundlage der Berechnung der Exponentialfunktionen auf andere Zahlenbasen (z. B. 10, e) dienen.
Der 8087-Befehlsvorrat enthält die Operation F2XM1 für die Berechnung der Funktion $Y = 2^x - 1$ für den Wertebereich $0 \leq X \leq 0{,}5$. Das Ergebnis ersetzt die Vorgabe in ST. Für die Berechnung von Logarithmusfunktionen ist der Befehl FYL2X geeignet. Aus den Argumenten X in ST und Y in ST(1) mit den Wertebereichen $0 < X < \infty$ und $-\infty < Y < +\infty$ berechnet FYL2X die Funktion $Z = Y * ld(X)$. Mit $Y = \log_g(2)$ kann damit der Logarithmus zur Basis g berechnet werden. Für höhere Genauigkeitsanforderungen kann die Funktion $Z = ld(X+1)$ mit dem Befehl FYL2XP1 berechnet werden.
Die wichtigsten der für die Umrechnung aller Standardfunktionen und ihrer vollen Argumentbereiche auf die 8087-Funktionen benötigten Konstanten werden durch spezielle Ladebefehle bereitgestellt:

```
FLDZ   ; ST(0) = 0
FLD1   ; ST(0) = 1
FLDPI  ; ST(0) = π
FLDL2T ; ST(0) = ld(10)
FLDL2E ; ST(0) = ld(e)
FLDLG2 ; ST(0) = lg(2)
FLDLN2 ; ST(0) = ln(2)
```

7.5.6 Prozessorsteuerbefehle
8087-Steuerbefehle dienen der Initialisierung, der Programmierung der Betriebsweise und der Interruptverarbeitung.
Mit FINIT wird der 8087 in der gleichen Weise, wie nach einem RESET-Signal initialisiert. Die Freigabe und Sperre von 8087-Interrupts ist mit den Befehlen FENI und FDISI möglich. Ein im Speicher bereitgestelltes 8087-Steuerwort nach Tafel 7.4 wird mit FLDCW in das 16-Bit-Steuerregister übernommen. Dabei wird die Speicheradresse in der für 2-Byte-Integer-Daten üblichen Form angegeben.
Sowohl das Statuswort als auch das Steuerwort können mit den Befehlen FSTSW und FSTCW in den Speicher geschrieben werden. Der Befehl FCLEX löscht alle Ausnahmemarkierungen, das Interruptanforderungs- und das BUSY-Bit im Statuswort.
Für die Interruptbehandlung existieren im 8087 leistungsfähige Befehle für die blockweise Auslagerung der 8087-Informationen in den Speicher und die entsprechenden Rückladeoperationen.
Die Befehle FSTENV und FLDENV dienen dem Abspeichern und Zurückschreiben von Status-, Steuer- und Tag-Wort und der Zeiger für den zuletzt bearbeiteten Speicheroperanden und für den letzten Befehl.
Noch leistungsfähiger ist das Befehlspaar FSAVE und FRSTOR, das zusätzlich die drei letzten 10-Byte-Stapeleintragungen ST, ST(1) und ST(2) abspeichert bzw. zurückschreibt.
Weitere Steuerbefehle FINCSTP und FDECSTP übernehmen das Inkrementieren und Dekrementieren des Registerstapelzeigers. Der Befehl FFREE setzt eine 'Leer'-Markierung im Tag-Wort für ein ausgewähltes Register.

7.5.7 Synchronisation der 8086-8087-Parallelarbeit
Die in einem gemeinsamen Befehlsstrom enthaltenen 8086- und 8087-Befehle werden von beiden Prozessoren parallel ausgewertet.
Die CPU kann noch während der Bearbeitung des 8087-Befehls durch den Arithmetikprozessor, nach der BUS-Freigabe, die Bearbeitung der nächsten 8086-Befehle übernehmen.
Vor der Verarbeitung des nächsten 8087-Befehls durch beide Prozessoren muß jedoch gesichert sein, daß der Arithmetikprozessor die Abarbeitung des vorherigen Befehls beendet hat. Das gilt auch, wenn ein folgender 8086-Befehl Ergebnisse des vorangegangenen 8087-Befehls verwertet.
Durch das Einfügen von 8086-WAIT-Befehlen kann die zeitliche Synchronisation erreicht werden.
Die schaltungstechnische Grundlage dieser Zeitsteuerung bildet die Anschaltung des BUSY-Ausgangs des 8087 an den $\overline{TEST}$-Eingang der CPU.
Ein WAIT-Befehl vor einem 8087-Befehl garantiert, daß die Abarbeitung dieses Befehls durch den 8086 erst beginnt, wenn der Arithmetikprozessor seine Verarbeitungsbereitschaft durch ein nichtaktives BUSY-Signal meldet. Um dieses abzusichern, erzeugen einige Assembler für die 8087-Befehle automatisch vorangestellte 8086-WAIT-Maschinenbefehle. Durch spezielle mnemonische Beschreibungen für einige 8087-Steuerbe-

fehle kann für diese die Hinzunahme des WAIT-Befehls unterdrückt werden. Zur Differenzierung von WAIT-Befehlen durch Assemblerprogramme existiert eine zusätzliche mnemonische Beschreibung FWAIT für den 8086-WAIT-Befehl. Die FWAIT-Befehle können durch eine Einstellung des Assemblerprogrammes annulliert werden (z. B. bei 8086-Emulatorprogrammen für die 8087-Befehle).

7.5.8 Beispielprogramm

Das Beispielprogramm berechnet den Logarithmus eines Argumentes zu einer beliebigen Basis. Das Argument wird in das Stackregister 1 und die Basis in den *Top of Stack* geladen. Das Ergebnis steht im *Top of Stack*.

```
;
; Übergabe:
;            TOS: Basis
;            ST1 : Argument
;            ST7 : muss frei sein
; Rückgabe:
;            TOS: Ergebnis
;         ST6, ST7 : frei
;         Die übrigen Registerinhalte
;         bleiben unverändert.

 cseg       0
log:
 call       logarithmieren
                  ; Logarithmieren
                  ; der Basis

; Logarithmieren des Arguments
;
 fxch       st1   ; TOS: Argument
                  ; ST1:log (basis)
 call       logarithmieren
 fdivdp     st1
 mov        byte ptr log-wahl, false
 ret
logarithmieren:
 fld1
 fxch       st1   ; TOS: Argumente, ST1: 1
 fYl2x            ; TOS: = ST1 * log2(TOS)
 ret
 dseg       0
log-wahl    db    0
false       equ   0
```

7.6 Der Input/Output-Mikroprozessor 8089

Im folgenden Abschnitt wird ein weiterer Mikroprozessor des 8086-Systems vorgestellt, der durch seine Multiprocessing-Mechanismen, DMA-Eigenschaften und einen auf Peripheriebedingungen spezialisierten Befehlssatz zur Steigerung der Leistungsfähigkeit des Systems beiträgt.

7.6.1 Entwicklung der Peripheriebaugruppen der Mikroprozessorsysteme (Bild 7.7)

Die erste Generation der Mikroprozessoren war durch mit TTL-Schaltkreisen aufgebaute Peripheriesteuerungen gekennzeichnet. In der nächsten Generation wurden Single-Chip Peripherie-Controller eingeführt, die durch Programmierung in der Anwendung komfortabler waren. Der Datentransfer von bzw. zur Peripherie wurde weiterhin von der CPU kontrolliert. Mit der Einführung von DMA-Schaltkreisen (Direct Memory Access) konnten schnelle Datentransferoperationen auch unabhängig von der CPU realisiert werden. In der weiteren Entwicklung wurde der DMA-Schaltkreis mit Eigenschaften eines Mikroprozessors versehen und zu einem spezialisierten Input-/Output-Prozessor vereinigt. Dieser bildet mit der zentralen CPU ein Multiprozessorsystem und entlastet durch Parallelarbeit die CPU von zeitaufwendiger Peripheriebedienung. Ein Input-/Output-Prozessor, der auf einem Chip die Eigenschaften eines DMA und Prozessors integriert, ist der Input-/Output-Prozessor (IOP) 8089.

7.6.2 Der I/O-Prozessor im 8086-System

Der I/O-Prozessor 8089 bildet mit der 8086/88-CPU ein leistungsfähiges Multiprozessorsystem. Bild 7.1 zeigt eine zentrale Verarbeitungseinheit mit den Prozessoren 8086, 8087 und 8089 ohne zusätzlichen Hardwareaufwand. Der 8086 arbeitet in dieser Konfiguration als Master und die anderen Prozessoren im „local"-mode als Slave. In einer anderen Systemkonfiguration (Bild 7.8) arbeiten zwei IOPs ohne 8086-CPU im sogenannten „remote"-mode, wobei einer als Master und der andere als Slave fungiert.

Die Statussignale $\overline{S0} \ldots \overline{S2}$ sind mit dem Bus-Controller 8288 verbunden und ergeben folgende Kodierung:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Bedeutung
0	0	0	Instruction fetch from I/O-Space
0	0	1	Data fetch from I/O-space
0	1	0	Data store to I/O-Space
0	1	1	not used
1	0	0	Instruction fetch from system space
1	0	1	Data fetch from system space
1	1	0	Data store to system space
1	1	1	no bus cycle run

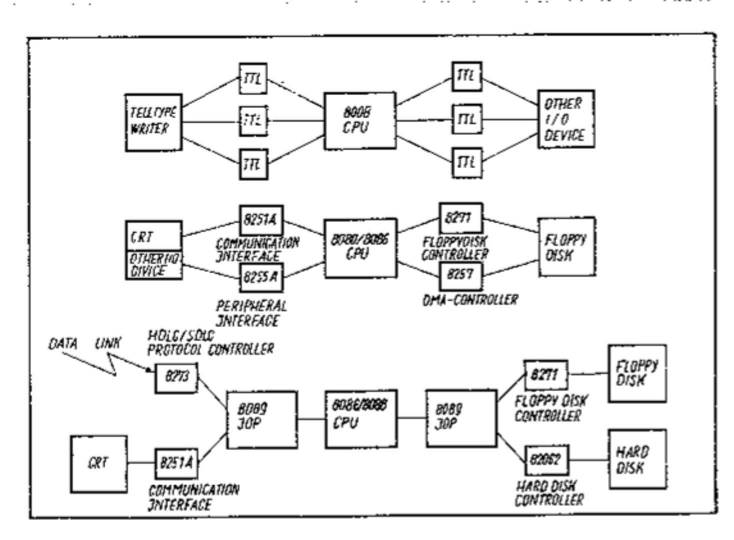
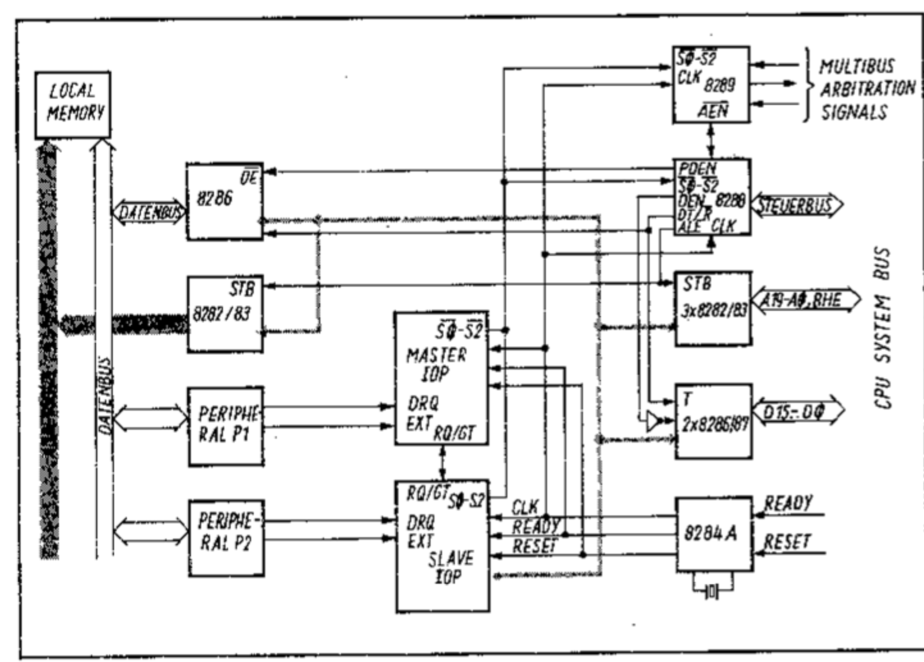


Bild 7.7 Entwicklung der Peripheriebaugruppen

Bild 7.8 Zweiprozessorsystem mit 8089 im „remote" Mode



Literatur

/1/ Dokumentation SCP 1700. VEB Robotron Elektronik Dresden
/2/ CP/M-86 Programmer's Guide. Digital Research 1981
/3/ CP/M-86 User's Guide. Digital Research 1981
/4/ Heckel, U.: Das Betriebssystem SCP 1700 EDV-Aspekte 6 (1987) 1, S. 18

wird fortgesetzt

Mikroprozessorsystem K 1810 WM 86

Hardware · Software · Applikation (Teil 6)

Prof. Dr. Bernd-Georg Münzer
(wissenschaftliche Leitung),
Dr. Günter Jorke, Eckhard Engemann,
Wolfgang Kabatzke, Frank Kamrad,
Hellfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock,
Sektion Technische Elektronik,
Wissenschaftsbereich Mikrorechentechnik/
Schaltungstechnik

Die Adreßleitungen A16 . . . A19 werden mit S3 . . . S6-Statussignalen zeitgemultiplext:

S6	S5	S4	S3	Bedeutung
1	1	0	0	DMA on channel 1
1	1	0	1	DMA on channel 2
1	1	1	0	non-DMA on channel 1
1	1	1	1	non-DMA on channel 2

7.6.2 Interne Architektur des IOP
Im Bild 7.9 ist die interne Architektur des I/O-Prozessors dargestellt.

CCU *Common Control Unit*
Alle IOP-Operationen (Befehle, DMA-Transfers usw.) setzen sich aus einer Reihe von internen Buszyklen zusammen, die durch die CCU koordiniert werden. Die CCU kontrolliert auch das Initialisieren des IOP.

ALU *Arithmetic-Logic Unit*
Die ALU kann arithmetische Operationen (Addition, Decrement, Increment) mit 8- oder 16-Bit breiten, nicht vorzeichenbehafteten, binären Zahlen ausführen, wobei das Ergebnis eine binäre Zahl mit maximal 20 Bit sein kann. Zum Befehlsvorrat der ALU gehören auch die logischen Operationen **AND, OR, NOT.**

Assembly/Disassembly Registers
Alle transferierten Daten werden über diese Register geführt. Im Falle eines Datentransfers zwischen zwei Bussen unterschiedlicher Breite fungieren diese Register zur Busanpassung, um zeitlich optimale Datenübertragung zu erreichen.

Beispiel
Quelle: 8 Bit breiter Bus
Ziel: 16 Bit breiter Bus

Für die Quelle werden zwei Lesezyklen durchgeführt, die gelesenen 2 Bytes werden zum 16 Bit Wort zusammengefügt und in einem Schreibzyklus zum Ziel übertragen.

IFU *Instruction Fetch Unit*
Diese Einheit kontrolliert mit Hilfe eines 1-Byte-Queue das Befehlsholen für das aktuelle Kanalprogramm. Bei einem 8-Bit breiten Systembus werden die Befehle byteweise geholt. Für den 16-Bit breiten Bus gibt es zwei Varianten, die in den Bildern 7.10 und 7.11 dargestellt sind.

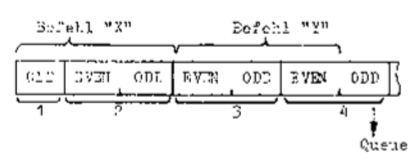


Bild 7.10 Befehlsholen angefangen bei EVEN-Adresse
Fetch Befehlsbyte
1 *Die ersten zwei Bytes von „X"*
2 *Das dritte Byte von „X" und das erste Byte von „Y", das im Queue zwischengespeichert wird*
3 *Das erste Byte von „Y" aus dem Queue: kein Buszyklus*
4 *Die letzten 2 Bytes von „Y"*

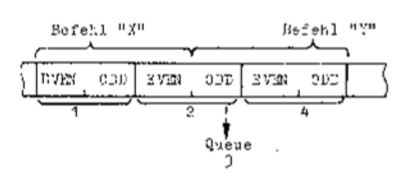


Bild 7.11 Befehlsholen angefangen bei ODD-Adresse
Fetch Befehlsbyte
1 *Das erste (ODD-adressierte) Byte (8-Bit-Zyklus) von „X"*
2 *Das zweite und dritte Byte von „X"*
3 *Das erste und zweite Byte von „Y"*
4 *Das dritte Byte von „Y" und das erste Byte vom nächsten Befehl, zwischengespeichert im Queue*

BIU *Bus Interface Unit*
Die BIU führt die Buszyklen aus, steuert den Transfer von Befehlen und Daten zwischen IOP und Speicher bzw. Peripherie.

Channels
Der IOP besitzt zwei unabhängige Kanäle, die sowohl ein eigenes Kanalprogramm, als auch einen DMA-Transfer ausführen können. Jeder Kanal besitzt eine eigene I/O-Steuereinheit, welche die Steuerung während des DMA-Transfers übernimmt. Mit dem DRQ-Signal (DMA-Request) kann der DMA-Transfer synchronisiert werden. Der Kanal wartet auf dieses Signal, bevor er den nächsten Schreib-/Lesezyklus durchführt. Mit dem EXT-Signal kann der laufende DMA-Transfer unterbrochen werden.

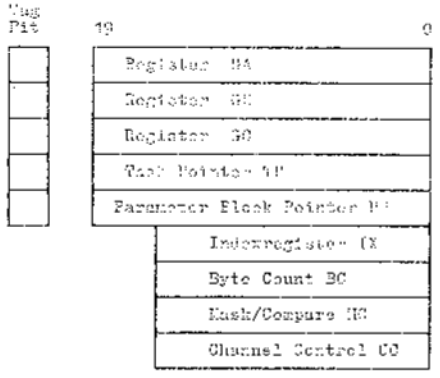


Bild 7.12 Registersatz des 8089

Kanalregisterplatz (Bild 7.12)
GA,GB: allgemeine Register im Kanalprogramm; Ziel oder Quelle im DMA-Transfer
GC: allgemeines Register im Kanalprogramm; zeigt im DMA-Transfer auf das erste Byte der Translationstabelle. Die bei der Quelle gelesenen Daten werden als nicht vorzeichenbehaftete Ziffern interpretiert, die zum Inhalt des GC-Registers addiert werden. Damit entsteht eine Adresse, die auf eines der 256 Bytes der Translationstabelle zeigt. Das Byte wird aus der Translationstabelle entnommen und zum Ziel übertragen (Bild 7.13).
PP: kann nur als Parameter-Block-Pointer genutzt werden und wird während der Initialisierung geladen.
TP: arbeitet als Befehlszähler; sollte im Kanalprogramm nicht verwendet werden.
IX: als allgemeines Register oder als Indexregister verwendbar.
BC: allgemeines Register oder Bytezähler im DMA-Betrieb.

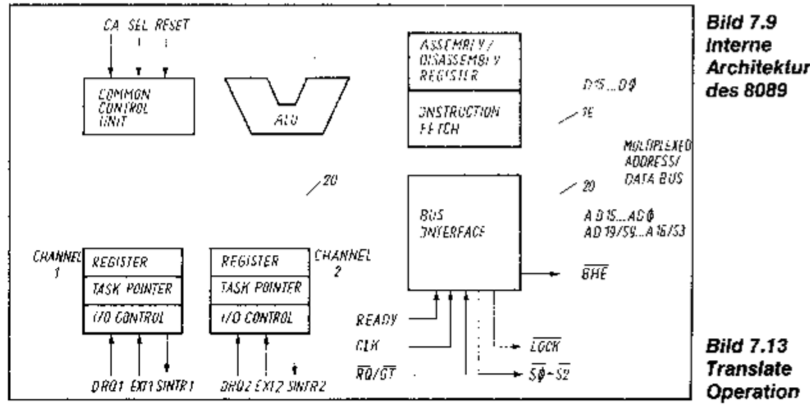


Bild 7.9 Interne Architektur des 8089

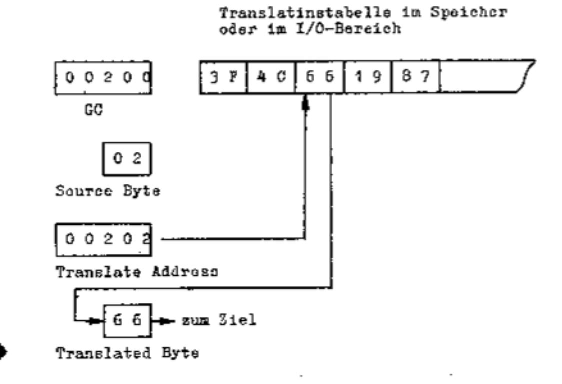


Bild 7.13 Translate Operation

MC: allgemeines Register, auch zum maskierbaren Vergleich verwendbar. Das Low-Byte wird mit dem Vergleichsbyte und das High-Byte mit der Maske geladen. Low in einer Bitposition bedeutet, daß das entsprechende Bit im Vergleichsbyte nicht berücksichtigt wird.
CC: Channel Control (Tafel 7.7)
Dieses Register wird im Kanalprogramm geladen. Der Inhalt des Registers ist in 10 Felder aufgeteilt, die die Informationen über den DMA-Transfer beinhalten.

Tafel 7.7 Channel Control Register

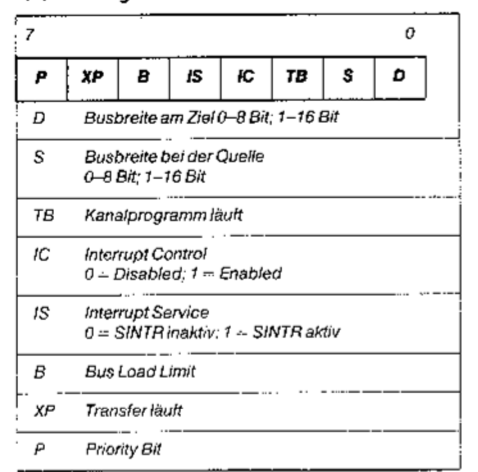
15					7				0
F	**TR**	**SYN**	**S**	**L**	**C**	**TS**	**TX**	**TBC**	**TMC**

F:	*Funktion*
	0 0 Port-zu-Port DMA
	0 1 Speicher-zu-Port-DMA
	1 0 Port-zu-Speicher-DMA
	1 1 Speicher-zu-Speicher-DMA
TR:	*Translation*
	0 DMA ohne dynamische Übersetzung
	1 mit Übersetzung
SYN:	*Synchronisation*
	0 0 ohne Synchronisation
	0 1 Synchronisation durch Quelle
	1 0 Synchronisation durch Ziel
	1 1 reserviert
S:	*Quelle:*
	0 GA zeigt auf die Quelle
	1 GB zeigt auf die Quelle
L:	*Lock*
	0 kein Lock während des DMA
	1 aktualisiere Lock während DMA
C:	*Chain*
	0 ohne Chain
	1 Chained Programm
TS:	*Abbruch nach einem DMA-Zyklus*
	0 kein Abbruch
	1 Abbruch nach einem Zyklus
TX:	*Abbruch durch externes Signal*
	0 0 kein Abbruch
	0 1 Abbruch Offset = 0
	1 0 Abbruch Offset = 4
	1 1 Abbruch Offset = 8
TBC:	*Abbruch wenn BC = 0*
	0 0 kein Abbruch
	0 1 Abbruch Offset = 0
	1 0 Abbruch Offset = 4
	1 1 Abbruch Offset = 8
TMC:	*Abbruch wenn ein Matchbyte gefunden*
	0 0 0 kein Abbruch
	0 0 1 Abbruch Offset = 0
	0 1 0 Abbruch Offset = 4
	0 1 1 Abbruch Offset = 8
	1 0 0 nichts
	Abbruch wenn kein Matchbyte gefunden
	1 0 1 Abbruch Offset = 0
	1 1 0 Abbruch Offset = 4
	1 1 1 Abbruch Offset = 8

TAG-Bit: (Bild 7.12)
Die Register GA, GB, GC und TP sind Pointer-Register, die Zeigeradressen für I/O- oder Speicherbereich beinhalten. Zu jedem der Register gehört ein TAG-Bit, welches zwischen den Bereichen unterscheidet:
TAG-Bit = 0 ⇒ Speicherbereich
TAG-Bit = 1 ⇒ I/O-Bereich

PSW *Program Status Word* (Tafel 7.8)
Jeder Kanal besitzt ein eigenes PSW, in dem der Zustand des Kanals abgespeichert wird. Dadurch ist es möglich, die Kanalprogramme zu suspendieren und dann durch Regenerierung des alten PSW neu zu starten. Das Kanalprogramm kann auf das PSW nicht zugreifen.

Tafel 7.8 Program Status Word



7							0
P	**XP**	**B**	**IS**	**IC**	**TB**	**S**	**D**

D	*Busbreite am Ziel 0–8 Bit; 1–16 Bit*
S	*Busbreite bei der Quelle 0–8 Bit; 1–16 Bit*
TB	*Kanalprogramm läuft*
IC	*Interrupt Control 0 = Disabled; 1 = Enabled*
IS	*Interrupt Service 0 = SINTR inaktiv; 1 = SINTR aktiv*
B	*Bus Load Limit*
XP	*Transfer läuft*
P	*Priority Bit*

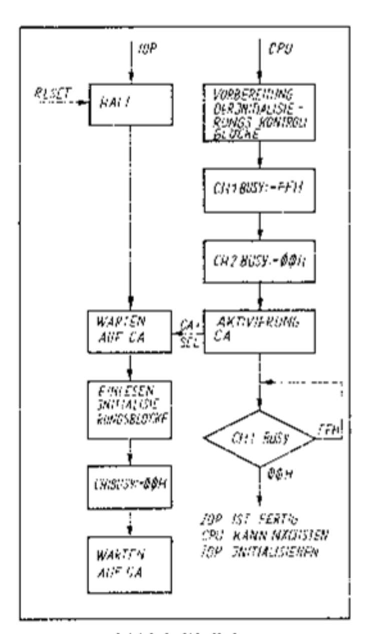


Bild 7.14 8089-Initialisierungssequenz

7.6.3 Initialisierung des IOP
Bevor die IOP-Kanäle die Steuerung der I/O-Geräte übernehmen, müssen sie initialisiert werden (Bild 7.14). Der IOP wird zum Einlesen seiner Initialisierungsblöcke durch die CA- und SEL-Signale (CA = Channel Attention, SEL = Auswahl des Kanals) aktiviert (Bild 7.1). Während der Initialisierung erfolgt die Master-Slave-Zuordnung:
- SEL = 0; Kanal 1; IOP programmiert als Master
- SEL = 1; Kanal 2; IOP programmiert als Slave

Die Kanäle des IOP können im I/O-Bereich oder im Speicherbereich der CPU plaziert werden. Im ersten Fall erfolgt das Aktivieren mit einem OUT- und im zweiten Fall mit einem MOV-Befehl.
In der Initialisierungssequenz liest der IOP die sich im Speicher befindenden Initialisierungsblöcke ein (Bild 7.15). Der Inhalt der residenten Anweisungen wird beim Programmieren der EPROMs festgelegt, der RAM-Teil des Initialisierungssteuerblockes muß vor dem IOP-Start durch ein CPU-Programm geladen werden.
- *System Configuration Pointer (SCP),* der mit dem SYSBUS-Byte die Systembusbreite festlegt (Bild 7.16) und mit SCB Segment Base und Offset den Anfang des System-Configuration-Blockes (SCB) angibt.
- *System Configuration Block (SCB)* In diesem Block befindet sich das SOC-Byte (Bild 7.17), mit dem die I/O-Busbreite und Request/Grant-Mode festgelegt wird. Mit dem CB Segment Base und Offset wird die Anfangsadresse des Channel-Control-Blockes angegeben.
- *Channel Control Block (CB)* In diesem Block wird jedem Kanal sein Channel Control Word CCW (Tafel 7.9) und die Fixierung des BUSY-Flagbytes mitgeteilt. Weiterhin bekommt jeder Kanal mit dem PB Segment Base und Offset die Adresse seines Parameter-Blockes.

– *BUSY* ist ein Flagbyte, wird vor der Initialisierung von der CPU mit 0FFH geladen und

Bild 7.15 Initialisierungssteuerblöcke

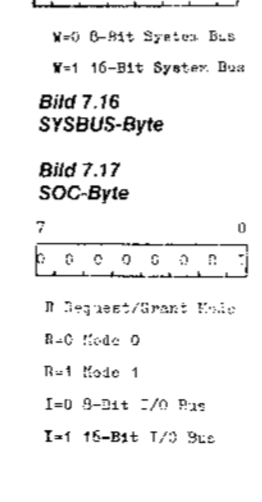
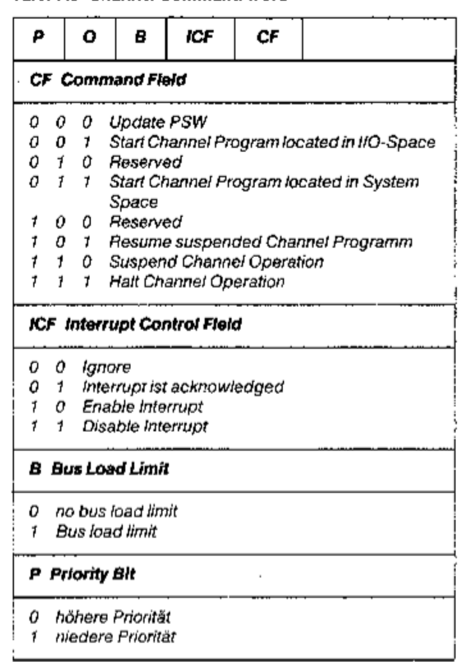


Bild 7.16 SYSBUS-Byte

Bild 7.17 SOC-Byte

Tafel 7.9 Channel Command Word



P	O	B	ICF	CF

CF Command Field

0	0	0	Update PSW
0	0	1	Start Channel Program located in I/O-Space
0	1	0	Reserved
0	1	1	Start Channel Program located in System Space
1	0	0	Reserved
1	0	1	Resume suspended Channel Programm
1	1	0	Suspend Channel Operation
1	1	1	Halt Channel Operation

ICF Interrupt Control Field

0	0	Ignore
0	1	Interrupt ist acknowledged
1	0	Enable Interrupt
1	1	Disable Interrupt

B Bus Load Limit

0	no bus load limit
1	Bus load limit

P Priority Bit

0	höhere Priorität
1	niedere Priorität

nach der Initialisierung vom IOP mit 00H überschrieben. Im Kanalprogramm wird das Byte vom IOP 0FFH gesetzt und am Ende mit 00H zurückgesetzt. Mit diesem Flag wird die Aufgabenzuweisung von der 8086-CPU gesteuert.

– *CCW Channel Command Word:* Nach der Initialisierung wird mit jeder CA-Aktivierung das im CCW kodierte Kommando für Kanal 1 oder 2 gelesen und ausgeführt. Mit dem P-Bit wird die Priorität des Kanals festgelegt. (Tafel 7.9)

– *PB: (Parameter Block)* beinhaltet die Startadresse des Kanalprogramms und dient auch zur Zwischenspeicherung von Kanalzustand (PSW) und Task Pointer (TP) mit seinem TAG-Bit, welches das Suspendieren und Neustarten von Kanalprogrammen ermöglicht.

7.6.4 Kanalprogramm

Nach der Initialisierungsphase kommt die Ausführungsphase, die mit einem OUT- oder MOV-Befehl gestartet wird. Die Kanalzuweisung erfolgt mit SEL (SEL = 0 Kanal 1; SEL = 1 Kanal 2). Nach Erkennen des CA-Signals führt der adressierte Kanal folgende Aktivitäten durch:

– BUSY-Flag setzen
– Einlesen des CCW vom Control Block (CB)
– Start und Ausführen der im CCW festgelegten Operation
(Bilder 7.18, 7.19, 7.20, 7.21)
– Rücksetzen des BUSY-Flags nach Programm- oder DMA-Transferende.

7.6.5 DMA-Transfer

Der IOP 8089 realisiert einen DMA-Wort-Datentransfer mit der Geschwindigkeit von 1,25 MByte/s.

Der DMA-Transfer wird vom Kanalprogramm aus mit der Anweisung XFER gestartet. Der danachfolgende Befehl wird noch abgearbeitet, und dann geht der IOP in die DMA-Phase über. Dieser zusätzliche Befehl kann zum Beispiel das letzte Steuerwort in einen Peripheriecontroller laden.

Im DMA-Transfer können eine Reihe von Abbruchbedingungen ausgelöst werden (siehe Channel Control Register und Bild 7.7). Nachdem eine Abbruchbedingung erkannt wurde, wird zum aktuellen Task Pointer der im CC-Register programmierte Offset 0, 4 oder 8 addiert. An diesen berechneten Adressen befinden sich LJUMP-Befehle (Long Unconditional Jump) zu Abbruchserviceprogrammen (Bild 7.22).

7.6.6 Interrupts

Jeder Kanal besitzt die Möglichkeit, über die Leitungen SINTR 1 und 2 eine Interruptanmeldung an die CPU zu senden. Die Interruptsperre bzw. -freigabe erfolgt durch das ICF-Feld des CCW (Tafel 7.9). Nachdem ein Kanal einen Interrupt ausgelöst hat, muß in der Interrupt-Serviceroutine die Interruptanforderung bestätigt und zurückgesetzt werden. Dies erfolgt mit dem Laden eines neuen CCW, in dem das ICF-Feld den logischen Wert „0,1" (interrupt acknowledge) trägt. Nachdem der Kanal das neue CCW empfangen hat, wird das „Interrupt Service Bit" im PSW und die SINTR-Leitung zurückgesetzt.

7.6.7 Konkurrierende Kanaloperationen

Beide Kanäle des IOP können zwar gleichzeitig aktiviert werden, jedoch die aktuelle Buszuweisung übernimmt die CCU in der Prioritätenfolge nach Tafel 7.10. Falls beide Kanäle Operationen mit identischer Priorität

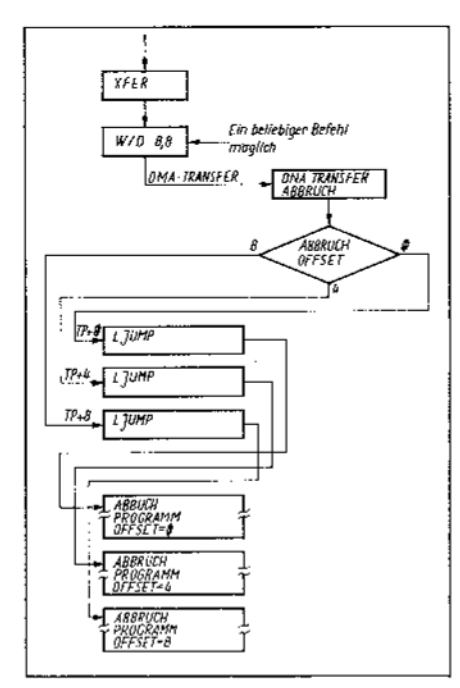


Bild 7.22 Abbruchverhalten des 8089

Tafel 7.10
Zusammenstellung der 8089-Aktivitätsprioritäten

Kanalaktivität	Priorität
DMA-Transfer	1
DMA-Abbruchsequenz	1
Kanalprogramm (Chained)	1
Kanalaktivierung (CA)	2
Kanalprogramm (not chained)	3
untätig (idle)	4

ausführen sollen, erfolgt die Entscheidung an Hand des Prioritätsbits in den PSWs (Tafel 7.8). Wenn diese Prioritätsbits wiederum gleich sind, wird von der CCU ein alternierendes Freigeben der Kanalaktivitäten vorgenommen. Die CCU kann die Kanalaktivitäten nur zu bestimmten Zeitpunkten (interleave boundaries) unterbrechen (Tafel 7.11).

Spezielle Bedingungen

Während LOCK aktiv ist, kann der DMA-Transfer nicht unterbrochen werden.

Kanalprogramm „chained" oder „not chained" kann unterbrochen werden, mit Ausnahme des TSL-Befehls.

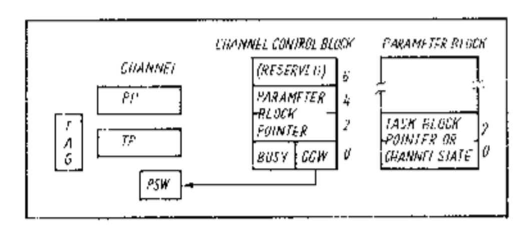
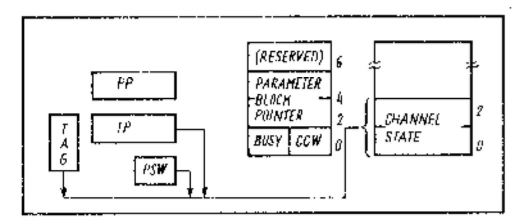


Bild 7.18 Update PSW

Bild 7.19 Start Program

Bild 7.20 Suspend Operation



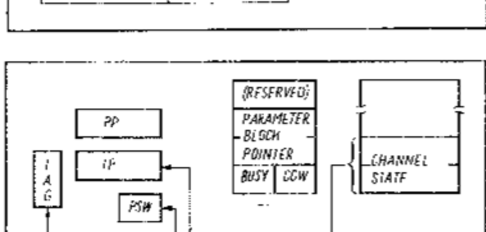


Bild 7.21 Resume Operation

Tafel 7.11 **Unterbrechungszeitpunkte (interleave boundaries)**

Kanal Aktivität	Priorität	Moment der Unterbrechung durch	
		DMA	Befehl
DMA	1	nach Bus-Zyklus	Bus-Zyklus
DMA-Abbruch Sequenz	1	interner Zyklus	keine
Kanal-Programm (chained)	1	interner Zyklus	Befehl
Channel Attention Sequenz	2	interner Zyklus	keine
Kanal Programm (not chained)	3	interner Zyklus	Befehl
untätig	4	zwei Takte	zwei Takte

7.6.8 *Multiprocessing-Eigenschaften*

Arbitrierungsprinzip ist mit dem des Arithmetik-Koprozessors 8087 vergleichbar und wird durch die $\overline{RQ}/\overline{GT}$-Signale realisiert. Die $\overline{RQ}/\overline{GT}$-Logik des IOP arbeitet in zwei verschiedenen Betriebsarten, die mit dem SOC-Byte (Bild 7.17) in der Initialisierung festgelegt werden:

Mode 0

$\overline{RQ}/\overline{GT}$-Mode 0 ist mit der $\overline{RQ}/\overline{GT}$-Logik der 8086/88-Prozessoren kompatibel (Bild 7.2). Wenn der IOP mit der 8086-CPU im Localmode arbeitet, ist er der Slave und die CPU der Master. In dem Fall, wenn zwei IOPs allein ein Multiprozessorsystem bilden, muß ein IOP während der Initialisierung als Master und der andere als Slave programmiert werden (vgl. Kapitel 7.6.3). Im Mode 0 hat der Masterprozessor keine Möglichkeit, den Bus vom Slave zurückzuverlangen.

Mode 1

Diese Betriebsart dient zum Arbitrieren der Zugriffe auf einen privaten (lokalen) I/O-Bus bei zwei parallel arbeitenden IOPs. In diesem Fall ist ebenfalls ein IOP als Master und der andere als Slave zu programmieren.
Die $\overline{RQ}/\overline{GT}$-Sequenz im Mode 1 lautet:
- Der anfordernde Prozessor sendet einen Request-Impuls.
- Der angeforderte Prozessor gibt den Bus mit einem Grant-Impuls ab.
- Wenn er die Buskontrolle wieder erhalten will, sendet er einen Request-Impuls zwei Takte nach dem Grant-Impuls.

Im Falle eines Multirechnersystems ist die Zugriffsarbitrierung auf gemeinsame Ressourcen mit dem Bus-Arbiter-Schaltkreis 8289 vorzunehmen.

7.6.9 *Assemblerbefehle des 8089*

- Datentransferbefehle

Datentransferbefehle dienen zum Datenaustausch (byte- oder wortweise) zwischen internen Kanalregistern und dem Speicher. Spezielle MOV-Befehle ermöglichen das Laden oder Abspeichern von Adressen und Tagbits in die Zeigerregister GA, GB, GC oder TP (Bild 7.12).

① **MOV** *destination, source*
Entsprechend Datenformat werden vier Gruppen unterschieden:

MOV	Move Word Variable
MOVB	Move Byte Variable
MOVI	Move Word Immediate
MOVBI	Move Byte Immediate

Da diese Befehle das Tagbit des entsprechenden Zeigerregisters setzen, eignen sie sich vor allem für das Laden der I/O-Adressen.

② **MOVP** *destination, source (move pointer)*
Der MOVP-Befehl transferiert eine physische Adresse und den Wert des Tagbits zwischen Zeigerregistern und Speicher (Bild 7.23).

Zeiger Register:

0	2	6	5	F	3	HEX
	0010	0110	0101	1111	0011	Binaer

Speicher:

100		101		102		Adresse
F	3	6	5	2	0	HEX
1111	0011	0110	0101	0010	0000	Binaer

Bild 7.23 MOVP-Befehl
○ Dieses Feld wird je nach Wert des Tag-Bit mit 0H (Tag-Bit = 0) oder mit 8H (Tag-Bit = 1) überschrieben.

③ **LPD** *destination, source (load pointer with doubleword)*
Der LPD-Befehl konvertiert einen sich im Speicher befindenden Doppelwortzeiger in eine 20 Bit breite physische Adresse und lädt sie in ein Zeigerregister (Bild 7.24).

100		101		102		103		
6	5	0	0	4	C	3	B	HEX
0110	0101	0000	0000	0100	1100	0011	1011	Binaer

Bild 7.24 LPD-Befehl
Wert des Doppelwortzeigers plaziert unter Adresse 100H:
Segment Base 3B4CH Offset 0065H
Im Zeigerregister geladene physische Adresse lautet: 3B525H

Das Tagbit des Zielregisters wird automatisch zurückgesetzt. Es werden zwei Typen vom LPD-Befehl unterschieden:

LPD	Load Pointer With Doubleword Variable
LPDI	Load Pointer With Doubleword Immediate

- Arithmetische Befehle

Die arithmetischen Befehle interpretieren alle Operanden als nicht vorzeichenbehaftete binäre Zahlen der Breite 8, 16 oder 20 Bit.
Der 8089 besitzt folgende Befehlsgruppen:
① **ADD** *destination, source*
Es werden folgende Additionsbefehle unterschieden, wobei das Ergebnis der Addition im Zieloperanden steht:

ADD	Add Word Variable
ADDB	Add Byte Variable
ADDI	Add Word Immediate
ADDBI	Add Byte Immediate

② **INC** *destination*
Inkrement des Ziels um 1.
INC Increment Word
INCB Increment Byte
③ **DEC** *destination*
Dekrement des Ziels um 1.
DEC Decrement Word
DECB Decrement Byte

- Logische und Bitmanipulationsbefehle

Die logischen Befehle des 8089 lauten: AND, OR, NOT, wobei die 4 höchsten Bits eines 20-Bit-Zielregisters undefiniert bleiben. Wenn ein Register das Ziel einer Byteoperation ist, werden die höherwertigen Bits 8–15 mit dem Wert des Bits 7 aufgefüllt.
① **AND** *destination, source*
OR *destination, source*
Zwei Operanden werden AND/OR-verknüpft, und das Ergebnis steht im Ziel.

AND/OR	Logical AND/OR Word Variable
ANDB/ORB	Logical AND/OR Byte Variable
ANDI/ORI	Logical AND/OR Word Immediate
ANDBI/ORBI	Logical AND/OR Byte Immediate

② **NOT** *destination, destination/source*
Der NOT-Befehl invertiert die einzelnen Bits des Operanden. Im Falle eines Operanden wird dieser vom Ergebnis überschrieben. Im Fall von zwei Operanden wird die negierte Quelle ins Ziel geschrieben (Ziel muß ein Register sein), der Ausgangswert bleibt erhalten. Der Befehl arbeitet sowohl mit Wort- als auch mit Byte-Operanden:

NOT	Logical NOT Word
NOTB	Logical NOT Byte

③ **SETB** *destination, bit-select*
Dieser Befehl wird zum Setzen von einzelnen Bits im Speicher benutzt. Der Bit-Select-Operand wählt das zu setzende Bit aus.
④ **CLR** *destination, bit-select*
Dieser Befehl führt ein Rücksetzen des ausgewählten Bits durch.

- Sprung und Aufrufbefehle

Das Register TP (Task Pointer) kontrolliert das sequentielle Abarbeiten der Befehle ähnlich einem Befehlszähler. Bei Programmverzweigungen wird zum TP die vorzeichenbehaftete Verschiebung (displacement) addiert. Diese Verschiebung kann eine 8-Bit- (short) oder 16-Bit-Zahl (long, Mnemonic L) sein. Das höchste Bit der Verschiebung ist das Vorzeichen (0 ⇒ positive, 1 ⇒ negative Verschiebung).
– **CALL/LCALL TPsave, target**
Dieser Befehl speichert den aktuellen TP-

Wert und das entsprechende Tagbit im TP-save-Operanden ab. Die Adresse der CALL-Routine wird durch Addition des aktuellen TP und des Target-Operanden berechnet.
Es wird empfohlen, den Rücksprung ins Hauptprogramm mit dem MOVP-Befehl zu realisieren, um so den alten geretteten TP-Wert wieder zu laden.
– **JMP/LJMP** *target*
Mit dem JMP-Befehl wird ein unbedingter Sprung ausgeführt, wobei der TP-Wert nicht gerettet wird.
– **JZ/LJZ** *source, target*
Der Sprung wird ausgeführt, wenn der Source-Operand Null ist, anderenfalls geht das Kanalprogramm zum nächsten Befehl über.
Wenn der Source-Operand ein 20-bit-Register ist, werden nur die 16 niederwertigen Bits ausgewertet.
– **JNZ/LJNZ** *source, target*
Der Sprung wird ausgeführt, wenn der Source-Operand ungleich Null ist.
– **JMCE/LJMCE** *source, target*
Dieser Befehl realisiert einen maskierbaren Vergleich zwischen dem Source-Byteoperanden und dem Inhalt des Mask-Compare-Registers. Das niederwertige Byte des MC-Registers ist das Vergleichsbyte und das höherwertige die Maske. Falls der Vergleich positiv ist, wird ein Sprung realisiert.
– **JMCNE/LJMCNE** *source, target*
Der Sprung wird ausgeführt, wenn das Vergleichsergebnis negativ ist.
– **JBT/LJBT** *source, bit-select, target*
Der Befehl testet das durch „Bit-Select" bestimmte Bit im Source-Operanden. Der Sprung wird ausgeführt, wenn das Bit gleich 1 ist.
– **JNBT/LJNBT** *source, bit-select, target*
Der Sprung wird ausgeführt, falls das selektierte Bit gleich 0 ist.
• Steuerbefehle
Die Steuerbefehle realisieren für das Kanalprogramm die Steuerung der Ausgänge LOCK und SINTR 1–2, das Initialisieren des DMA-Transfers oder das Festlegen der logischen Busbreite.
Beispiele:
– **TSL** *destination, set-value, target*
Mit diesem Befehl kann der Zugriff auf geteilte Ressourcen in einem Multiprozessorsystem gesteuert werden (Bild 7.25). Der Befehl aktiviert das LOCK-Signal, das zum Beispiel mit dem Busarbiter 8289 verbunden werden kann und veranlaßt den Busarbiter, während aktivem LOCK den Bus keinem anderen Master abzugeben. Der TSL-Befehl testet und setzt mit „set-value" die mit „destination" adressierte Speicherzelle. Der Befehl kann zur Implementierung einer Semaphorvariable dienen, mit der der Zugriff auf geteilte Ressourcen eines Multiprozessorsystems gesteuert wird. Wenn der „target"-Operand die Adresse des TSL-Befehls selbst ist, befindet sich das Kanalprogramm so lange in der TSL-Test-Schleife, bis „destination" den Wert 00H besitzt.
– **WID** *source-width, dest-width*
Der Befehl verändert die Bits 0 und 1 des PSW und legt somit die logische Busbreite der Quelle und des Zieles bei einem DMA-Transfer fest.

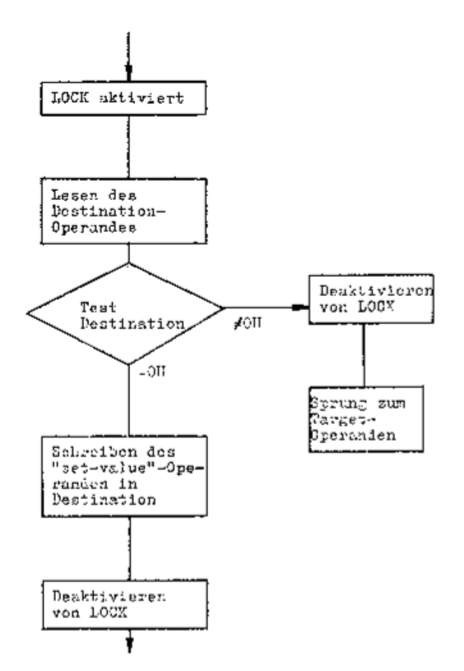


Bild 7.25 TSL-Befehl

Nach dem RESET sind die logischen Busbreiten undefiniert, so daß vor dem ersten Transfer der WID-Befehl ausgeführt werden muß.
– **XFER**
Mit diesem Befehl wird der DMA-Transfer nach dem dem XFER folgenden Befehl gestartet.
– **SINTR**
Dieser Befehl setzt das Interrupt-Service-Bit im PSW und aktiviert die SINTR-Ausgänge, falls das Interrupt-Control-Bit des PSW gesetzt ist.
– **NOP**
Leerbefehl
– **HLT**
Die Arbeit des Kanalprogramms wird angehalten, und das Kanal-BUSY-Byte wird zurückgesetzt.
• Beispielprogramm

LPDI GA, 1E00:0000H	; Anfangsadresse des Quellblockes für DMA-Transfer
LPDI GB, 1E00:8000H	; Anfangsadresse des Zielblockes für DMA-Transfer
MOVI BC, 0AH	; Anzahl der zu transferierenden Bytes
MOVI CC, 0C208H	; Laden des Channel-Control-Registers
XFER	; Start DMA
WID 16,16	; Quelle: 16 Bit breit Ziel: 16 Bit breit
HLT	

Das Beispielprogramm realisiert die Anfangsinitialisierung des IOP für einen Datentransfer von 10 (0AH) Bytes zwischen zwei mit den Registern GA und GB festgelegten Speicherblöcken. Die Startadresse des Programms lautet:
2000:100H
Das CC-Register wird mit dem Wert C208H geladen, was im Detail folgende Informationen für den IOP trägt:
– kein maskierbarer Vergleich
– Abbruch bei BC = 0; Offset 0
– keine Wirkung vom EXT-Signal
– kein Abbruch nach einem Transferzyklus
– „no chaining"
– LOCK aktiv während des Transfers
– GA adressiert die Quelle
– keine Synchronisation (DRQ unwirksam)
– keine Translate-Operation
– Speicher-Speicher-Transfer
Damit das Kanalprogramm ausgeführt werden kann, müssen die Initialisierungssteuerblöcke im Speicher entsprechend geladen werden (Bild 7.15).

8. Überblick zum System 80286

Dieser Abschnitt gibt einen Ausblick auf das 16-Bit-Mikroprozessorsystem 80286 als eine Weiterentwicklung des Systems 8086 /1/, /2/. Im Mittelpunkt stehen dabei die neuen Eigenschaften der CPU 80286, die dem Prozessor im Vergleich zum 8086 eine wesentlich höhere Leistungsfähigkeit verleihen:
• komplexere CPU-Architektur
• integrierte Speicherverwaltungseinheit
• physischer Adreßraum von 16 MByte, virtueller Adreßraum von 1 GByte je Task
• mehrere Speicherschutzfunktionen
• Unterstützung der Taskverwaltung
Diese Leistungsmerkmale sind besonders auf die Belange von Multiuser- und Multitask-Systemen ausgerichtet.

8.1 CPU-Architektur

Die CPU 80286 besteht aus vier Funktionseinheiten (Bild 8.1).
Die **Bus-Unit** (BU) stellt die Schnittstelle zum externen 80286-Bus her. Der 16-bit-Datenbus und der 24-Bit-Adreßbus sind getrennt herausgeführt, wodurch auch ein zeitliches Überlappen der Buszyklen möglich wird. Die Adresse für den nächsten Buszyklus wird bereits ausgegeben, bevor der aktuelle Buszyklus beendet ist. Dieses sogenannte Pipelined-Address-Timing am CPU-Bus des 80286 kann ausgenutzt werden, um bei einer hohen Busbandbreite auch relativ große Speicherzugriffszeiten zuzulassen.
Die BU des 80286 organisiert, genau wie die des 8086, ein vorausschauendes Befehlsholen. Die Befehlsbytes werden in eine 6 Byte tiefe Warteschlange (Prefetch Queue) eingespeist.
Zwischen BU und **Execution Unit** (EU) liegt noch eine zusätzliche Funktionseinheit, die **Instruction Unit** (IU). Diese dekodiert die von der BU kommenden Befehlsbytes und verwaltet eine weitere Warteschlange (Instruction-Queue), in der maximal drei dekodierte Befehle vor ihrer Abarbeitung in der EU zwischengespeichert werden.
Die **Address-Unit** (AU) beinhaltet eine leistungsfähige Speicherverwaltungseinheit, deren Funktionen im Abschnitt 8.3. näher erläutert werden.
Alle vier Funktionseinheiten arbeiten parallel zueinander, wodurch ein hoher Datendurchsatz erreicht wird.
Der Registersatz der 80286-CPU (Bilder 8.2 bis 8.4) enthält neben den von der 8086-CPU bekannten Hauptregistern, Base- und Indexregistern, dem Instruction-Pointer und dem Flagregister eine Reihe zusätzlicher Register, die im folgenden beschrieben werden.

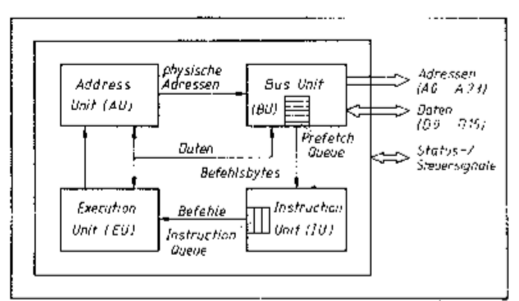
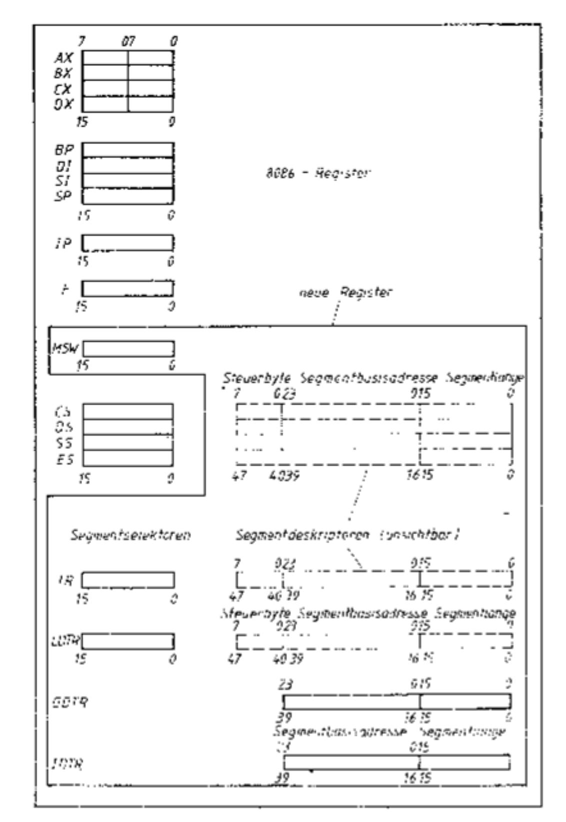


Bild 8.1 Blockschaltbild der CPU 80286

Bild 8.2 Registersatz der CPU 80286



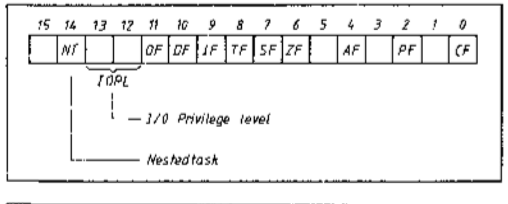


Bild 8.3 Flagregister der CPU 80286

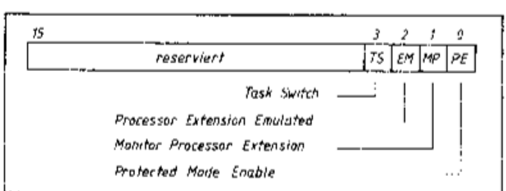


Bild 8.4 Maschinenstatuswort der CPU 80286

8.2 Betriebsarten

Die CPU 80286 kann in zwei Betriebsarten arbeiten: **Real-Address-Mode** (8086-Mode) und **Protected-Mode** (Protected-Virtual-Address-Mode).

Der Befehlssatz des 80286 ist voll abwärtskompatibel zu dem des 8086. Er enthält sämtliche 8086-Befehle und etwa 25 Ergänzungen. Einige dieser zusätzlichen Befehle (z. B. Multiplikationen mit Direktoperand, Block-Ein-/Ausgabe, ...) sind bereits im Real-Address-Mode verwendbar, andere dienen der Speicher- bzw. Taskverwaltung im Protected-Mode.

Nach RESET arbeitet der Prozessor im Real-Address-Mode und wirkt dabei genau wie eine 8086-CPU, das heißt, er unterstützt einen Speicherbereich von 1 MByte. Ein Unterschied besteht jedoch in der Arbeitsgeschwindigkeit. Erstens kann der 80286 mit einer höheren Taktfrequenz betrieben werden (je nach Typ maximal 6 bis 20 MHz), und zweitens erreicht er wegen des verbesserten Pipelinings bei gleicher Taktfrequenz einen etwa um den Faktor 2,5 höheren Datendurchsatz.

Das bedeutet, 8086-Maschinenprogramme sind auf einem 80286-Rechner im Real-Address-Mode ohne Änderungen lauffähig, abgesehen von Zeitbedingungen und der Hardwareumgebung.

Die volle Leistungsfähigkeit erreicht die 80286-CPU im Protected-Mode. Die Umschaltung in diese Betriebsart erfolgt durch das Setzen des Bit *Protected-Mode-Enable* im Maschinenstatuswort (Bild 8.4). Vorher sind jedoch einige Initialisierungen (z. B. Segmentdeskriptor-Tabellen) erforderlich. Eine Rückkehr aus dem Protected-Mode in den Real-Address-Mode ist nur durch RESET möglich.

Im Protected-Mode unterstützt die 80286-CPU einen physischen Adreßraum von 16 MByte (2^{24} Byte) und einen virtuellen Adreßraum von 1 GByte (2^{30} Byte) je Task. Der E/A-Adreßraum umfaßt 64 KByte.

8.3 Speicherverwaltung im Protected-Mode

Die 80286-CPU besitzt eine interne Speicherverwaltungseinheit, welche die Aufgabe hat, die im Programm verwendeten virtuellen Adressen auf den physischen Adreßbereich abzubilden. Gleichzeitig werden eine Reihe von Speicherschutzfunktionen realisiert.

Auch im Protected-Mode wird das Prinzip der Segmentierung verwendet, das heißt, der Adreßraum des 80286 ist in Segmente aufgeteilt, deren Länge von 1 Byte bis 64 KByte variabel ist.

8.3.1 Segment-Deskriptoren

Eine 24-Bit-Speicheradresse wird im Protected-Mode durch lineare Addition von einer 24-Bit-Segment-Basisadresse und einem 16-Bit-Offset gebildet. Der wesentliche Unterschied zum Real-Address-Mode liegt in der neuen Bedeutung der Segmentregister CS, DS, SS und ES. Diese Register enthalten nicht mehr die Segment-Basisadresse, sondern einen Segmentselektor. Deshalb heißen sie im Protected-Mode auch Selektorregister. Wie Bild 8.5 zeigt, dienen Bit 15...3 eines solchen Segmentselektors als Segmentdeskriptor-Index. Das ist ein Index für den dazugehörigen Segmentdeskriptor, der sich innerhalb einer im Speicher angelegten Segmentdeskriptor-Tabelle befindet. Dieser Segmentdeskriptor enthält nun die 24 Bit breite Segment-Basisadresse, aus der durch Addition mit dem 16-Bit-Offset die physische Speicheradresse (A0... A24) gebildet wird. Bild 8.6 veranschaulicht diesen Vorgang.

Der Umweg über die Segmentdeskriptor-Tabelle erwirkt einige Vorteile. Bild 8.7 zeigt den Aufbau eines Segmentdeskriptors. Er hat eine Länge von 8 Byte und enthält neben der Segment-Basisadresse noch die Länge des Segments und ein Steuerbyte, das den Typ des Segments und die Zugriffsrechte definiert. In den Bildern 8.8 und 8.9 sind die Formate der Steuerbytes für Daten- und Codesegmente angegeben. Die Angaben im Steuerbyte ermöglichen mehrere Speicherschutzfunktionen. Bei jedem Speicherzugriff wird automatisch die Einhaltung der festgelegten Segmentgrenzen (Länge) und Zugriffsrechte überprüft. Verletzungen dieser Bedingungen lösen spezielle Interrupts (Exceptions) aus.

Um die Speicherzugriffe durch die beschriebenen Vorgänge nicht zu verlangsamen, werden die aktuellen Segmentdeskriptoren im Prozessor zwischengespeichert. Im Protected-Mode bestehen die vier Segmentregister aus den Segmentselektor-Registern (CS, DS, SS, ES) und zusätzlich je einem unsichtbaren, 48 Bit breiten Segmentdeskriptor-(Cache-)Register (Bild 8.2). Dieses enthält eine Kopie des vom jeweiligen Segmentselektor ausgewählten Segmentdeskriptors.

Die Segmentselektoren werden durch die vom 8086 bekannten Befehle wie LDS, POP ES, JMPF (intersegment) beeinflußt. Im Protected-Mode lösen diese Befehle automatisch das Kopieren des neuen Segmentdeskriptors vom Speicher in das unsichtbare CPU-Cache-Register aus. Damit stehen der internen Speicherverwaltungseinheit während der folgenden Zugriffe auf dieses Segment die notwendigen Angaben zur Verfügung.

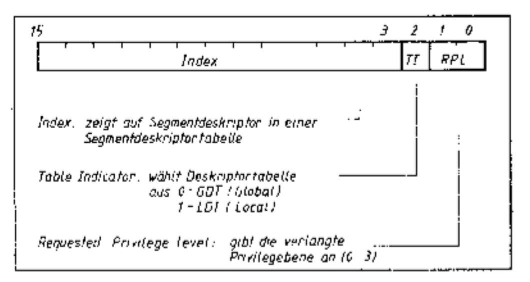


Bild 8.5 Aufbau eines Segmentselektors

7	0 7	0	
muß 0 sein, reserviert für System 80386			+6
Steuerbyte	Basisadresse A23-A16		+4
Basisadresse A15-A0			+2
Segmentlänge (Limit)			0

Bild 8.7 Segmentdeskriptor

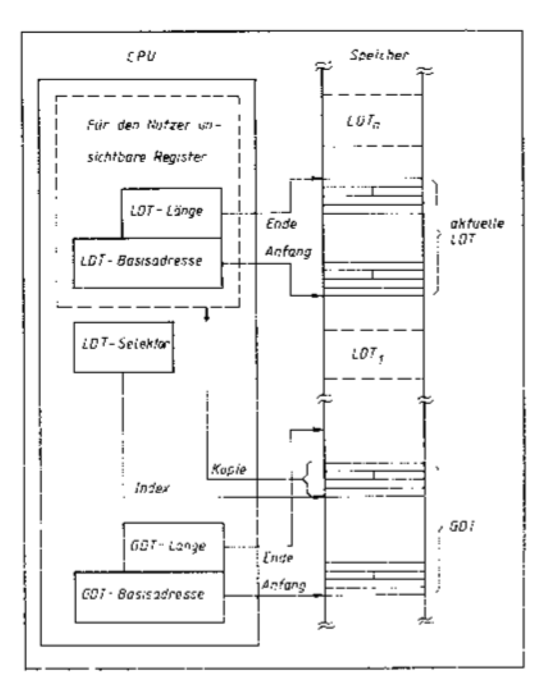


Bild 8.10 Globale und lokale Deskriptortabellen

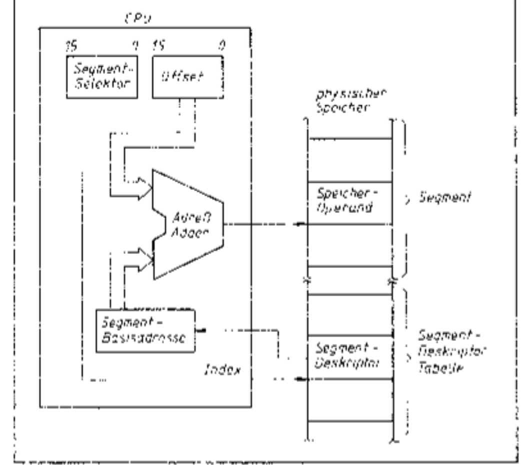


Bild 8.6
Speicheradressierung im Protected-Mode

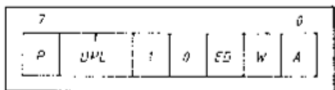


Bild 8.8 Steuerbyte
eines Datensegment-Deskriptors
Bit 7: P (Present) 1-Segment ist im Speicher vorhanden
Bit 6,5: DPL (Descriptor Privilege Level)
Privilegstufe 0...3
Bit 4 = ED (Expansion Direction)
0-Ausdehnung nach oben
Offset ≤ Limit
1-Ausdehnung nach unten
Offset > Limit (z. B. Stack)
Bit 1: W (Writeable) 0 – Read Only
1 – Read/Write
Bit 0: A (Accessed) 1 – Segmentdeskriptor wurde bereits benutzt

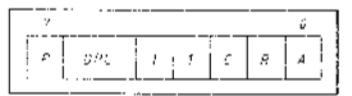


Bild 8.9 Steuerbyte
eines Codesegment-Deskriptors
Bit 4 = 1, Bit 3 = 1:
Kennzeichen für Codesegment
Bit 2: C (Conforming) 1-Segment paßt sich der aufrufenden Privilegebene an
Bit 1: R (Readable)
0-Segment ist nur ausführbar
1-Segment ist ausführbar und lesbar

8.3.2 Deskriptortabellen

• Globale und lokale Deskriptortabellen
80286-Programme werden in Programmteile, sogenannte Tasks, mit getrennten Speicherzugriffsrechten aufgeteilt.
Die Segment-Deskriptoren aller Speichersegmente, die von einer Task benutzt werden, sind in zwei Deskriptortabellen enthalten (Bild 8.10). Die **Global-Deskriptor-Table** (GDT) enthält die Deskriptoren der von allen Tasks nutzbaren globalen Segmente. Die **Local-Descriptor-Table** (LDT) enthält Deskriptoren von privaten Segmenten einer Task. Die Auswahl zwischen beiden Deskriptortabellen wird durch das TI-Bit im Segmentselektor getroffen (vergleiche Bild 8.5).
Mit den 13 Bit des Segmentdeskriptor-Index werden $2^{13} = 8192$ Segmentdeskriptoren (je 8 Byte) in einer Segmentdeskriptor-Tabelle adressiert. Somit stehen einer Task maximal 2^{14} Speichersegmente (8192 globale und 8192 lokale) mit einer Länge von je 64 KByte (2^{16} Byte) zur Verfügung. Daraus ergibt sich ein virtueller Adreßraum von einem Gigabyte (2^{30} Byte) je Task.
Die Deskriptortabellen sind selbst Speichersegmente. Ihre Lage innerhalb des physischen Speichers wird in entsprechenden CPU-Registern festgelegt (vergleiche Bild 8.2). Das **Global-Descriptor-Table-Register** (GDTR) enthält die Segment-Basisadresse (24 Bit) im absoluten Speicherraum und die Länge (16 Bit) der GDT. Das GDTR kann mit dem Load-Befehl LGDT geladen werden.
Das **Local-Descriptor-Table-Register** (LDTR) ist genau wie die oben beschriebenen Segmentregister aufgebaut. In einem sichtbaren 16-Bit-LDT-Selektor vom Format nach Bild 8.5 wird durch den Index die Lage des LDT-Deskriptors in der GDT festgelegt. In einem verdeckten 40-Bit-Cache-Register wird eine Kopie des aktuellen LDT-Deskriptors mitgeführt. Mit dem Befehl LLDT wird nur der Selektor der aktuellen LDT geladen, der mit dem Index zum Deskriptor der aktuellen LDT zeigt. Das Kopieren des entsprechenden Segmentdeskriptors, also Steuerbyte, LDT-Basisadresse und -Länge, in die CPU erfolgt dann automatisch. Wie in Bild 8.7 dargestellt, kann es mehrere LDT geben, deren Deskriptoren sich in der GDT im Speicher befinden.

• Interrupt-Deskriptortabelle
Die **Interrupt-Descriptor-Table** (IDT) ist vergleichbar mit der Interrupt-Tabelle des 8086. Sie enthält sogenannte Gate-Deskriptoren (je 8 Byte) für maximal 256 verschiedene Interrupts.
Basisadresse zur Lage der IDT im Speicherraum und Länge der IDT können mit dem Befehl LIDT in das **Interrupt-Descriptor-Table-Register** (IDTR) geladen werden (Bild 8.2).

8.4 Privilegkonzept

Die CPU 80286 gestattet nicht nur eine einfache Trennung von Systemprogrammen und Anwenderprogrammen, sondern unterstützt vier Software-Privilegebenen. Diese sind von 0 bis 3 numeriert, dabei ist Ebene 0 am meisten privilegiert und Ebene 3 am wenigsten (Bild 8.11).
Vom Privilegkonzept des 80286 sollen hier nur die wichtigsten Prinzipien erläutert werden.
Jedem Segment wird eine der vier Privilegebenen zugeordnet und als Descriptor-Privilege-Level (DPL) im Steuerbyte des Segmentdeskriptors kodiert (Bilder 8.8 und 8.9).
Die Privilegebene des aktuell ausgeführten Programms, **Current-Privilege-Level** (CPL), wird im RPL-Feld (vergleiche Bild 8.5) des aktuellen Codesegment-Selektors vorgegeben, also in Bit 1 und 0 des CS-Registers. Innerhalb einer Task muß CPL also nicht konstant bleiben, sondern kann sich, abhängig vom gerade ausgeführten Programm (Codesegment), ändern.
Für Datenzugriffe gilt nun die Regel, daß Programme nur auf Datensegmente der gleichen oder einer niederen Privilegebene zugreifen können. Es muß also gelten:

$$\text{CPL des aktuellen Codesegments} \leq \text{DPL des Datensegment-Deskriptors}$$

Ein numerisch kleinerer Wert entspricht einer höheren Ebene.
Bei Codezugriffen muß normalerweise die Regel

$$\text{CPL des aktuellen Codesegments} = \text{DPL des Ziel-Codesegment-Deskriptors}$$

eingehalten werden, das heißt, innerhalb einer Task können mit JMPF (intersegment) oder CALLF (intersegment) nur Codesegmente der gleichen Privilegebene aufgerufen

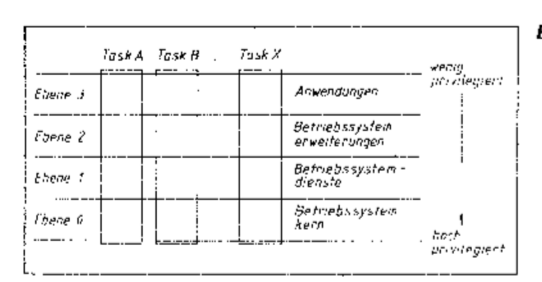


Bild 8.11 Privilegebenen

werden. Allerdings sind folgende zwei Ausnahmen gestattet:

- Codesegmente mit der Eigenschaft *conforming*, zum Beispiel Module des Betriebssystems, können auch von weniger privilegierten Programmen, z. B. Anwenderprogrammen, aufgerufen werden (vergleiche Bild 8.8). Sie passen sich dabei der aufrufenden Privilegebene an.
- Für den kontrollierten Übergang zwischen den Privilegebenen gibt es spezielle Deskriptoren, sogenannte CALL-Gates. Ein solcher Deskriptor (ähnlich wie in Bild 8.7) enthält den Selektor des Ziel-Codesegments (also den Verweis auf den entsprechenden Codesegment-Deskriptor) und den Offset des Eintrittspunktes in dieses Codesegment. Für den Aufruf eines CALL-Gate gelten die Bedingungen:

CPL des aktuellen Codesegments ≦ DPL des CALL-Gate-Deskriptors

CPL des aktuellen Codesegments ≦ DPL des Ziel-Codesegment-Deskriptors

Zur Durchsetzung des Privilegkonzeptes gibt es weiterhin einige privilegierte Befehle, die nur von Programmen der Privilegebene 0 ausgeführt werden dürfen. Dazu gehören zum Beispiel die Ladebefehle für die CPU-Register IDTR, GDTR und LDTR.
Ein- und Ausgabebefehle sind ebenfalls privilegiert. Die niedrigste Privilegebene, in der diese Befehle noch ausgeführt werden dürfen, wird als **I/O-Privilege-Level** (IOPL) bezeichnet und kann durch Bit 13 und 12 des Flagregisters festgelegt werden (Bild 8.3). Es muß also gelten:

CPL des aktuellen Codesegments ≦ IOPL

Die Einhaltung aller genannten Regeln wird von der CPU automatisch überprüft. Verletzungen lösen spezielle Status-Interrupts (Exceptions) aus.

8.5 Taskverwaltung

Ein weiteres wichtiges Leistungsmerkmal der 80286-CPU im Protected-Mode ist die interne Taskverwaltung.
Diese darf jedoch nicht als ein auf dem Chip installiertes Multitask-Betriebssystem verstanden werden. Die CPU erledigt nur die bei jedem Taskwechsel notwendigen Grundoperationen, das heißt, das Retten des alten und Laden des neuen CPU-Inhaltes. Die eigentliche Taskverwaltung, also die Entscheidung, wann welcher Taskwechsel erfolgt, muß nach wie vor softwaremäßig vorgenommen werden.

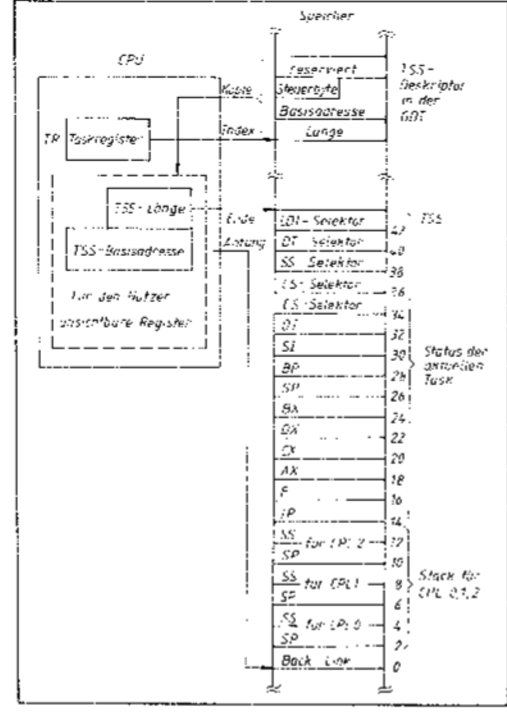


Bild 8.12
Task-State-Segment und TSS-Deskriptor

8.5.1 Task-State-Segment

Zu jeder Task gehört ein **Task-State-Segment** (TSS; Bild 8.12). Das ist ein Speicherbereich, in dem der Abarbeitungsstand der jeweiligen Task festgehalten wird. Dazu gehören Registerinhalte, Adreßbereiche (Segmentselektoren) und ein Verweis auf die vorhergehende Task (Back-Link). Die aktuelle Task wird durch ein spezielles Selektor-Register, das Task-Register TR, ausgewählt (vergleiche Bild 8.2). Dieses Register enthält einen Selektor, der nach dem bekannten Prinzip auf einen TSS-Deskriptor innerhalb der GDT verweist. Der Deskriptor enthält Basisadresse, Länge und Steuerbyte des aktuellen TSS. Diese Informationen werden auch hier in einem verdeckten CPU-Register mitgeführt.

8.5.2. Taskumschaltung

Eine Taskumschaltung erfolgt nun durch den Ansprung des entsprechenden Task-State-Segment mit einem Befehl JMPF (intersegment) oder CALLF (intersegment) oder über ein Task-Gate. Daraufhin werden automatisch (vergleiche Bild 8.12)
– alle Register in das aktuelle TSS gerettet
– das Task-Register mit dem neuen Selektor geladen
– der TSS-Deskriptor in den unsichtbaren Teil des Task-Registers kopiert
– die CPU-Register mit den Werten aus dem neuen TSS geladen
– die Abarbeitung der neuen Task dort fortgesetzt, wo sie zuletzt unterbrochen wurde.
Bei der Taskumschaltung wird außerdem in das Back-Link-Feld des neuen TSS der Selektor des alten TSS (alter TR-Inhalt) eingetragen. Das ist zum Beispiel dann von Bedeutung, wenn die Taskumschaltung von einem Interrupt ausgelöst wurde. Als Kennzeichen dafür dient das Bit NT (Nested-Task) im Flagregister (Bild 8.3). Bei gesetztem NT-Flag wird der Befehl IRET den Back-Link nutzen, um wieder eine Umschaltung auf die alte, durch den Interrupt unterbrochene Task auszulösen.

8.5.3 Task-Gates

Alle Task-State-Segment-Deskriptoren müssen in der GDT stehen. Damit Tasks auch über die LDT bzw. durch Interrupts auch über die IDT aufgerufen werden können, gibt es sogenannte Task-Gates. Das sind Vermittler-Deskriptoren, ähnlich den CALL-Gates, die außer dem Steuerbyte nur den Selektor des dazugehörigen TSS-Deskriptors enthalten. Task-Gate-Deskriptoren können in der GDT, der LDT und der IDT stehen und dienen dort als Verweis auf einen TSS-Deskriptor und damit auf ein Task-State-Segment.

Literatur
/1/ The 8086 Family User's Manual 1980
/2/ User Manual Intel 1979
/3/ 8087 Numeric Data Coprocessor. Intel Corporation 1986
/4/ Mikrokomputery szesnastobitowe, 6. Szkola Mikroprocesorowa, 3.–5. Dezember 1984. Polskie Towarzystwo Informatyczne Centrum Szkolenia Informatycznego ZETO, Łodz
/5/ Microsystem Components Handbook. Microprocessors Volume I. Intel 1986
/6/ Vieillefond, C.: Programmierung des 80286. Düsseldorf: SYBEX-Verlag 1987

wird fortgesetzt

Mikroprozessorsystem K 1810 WM 86

Hardware · Software · Applikation (Teil 7)

Prof. Dr. Bernd-Georg Münzer (wissenschaftliche Leitung), Dr. Günter Jorke, Eckhard Engemann, Wolfgang Kabatzke, Frank Kamrad, Hellfried Schumacher, Tomasz Stachowiak
Wilhelm-Pieck-Universität Rostock, Sektion Technische Elektronik, Wissenschaftsbereich Mikrorechentechnik/Schaltungstechnik

8.6. Systemschaltkreise

Die CPU 80286 ist ein VLSI-Schaltkreis mit etwa 130000 integrierten Transistoren im 68poligen Gehäuse.
Adreß- und Datenbus des Prozessors sind getrennt herausgeführt, belegen also 24 bzw. 16 Anschlüsse. Weitere 17 Pins führen Steuer- und Statussignale, ähnlich denen des 8086. Die restlichen Anschlüsse sind entweder nicht beschaltet oder dienen zur Spannungsversorgung. Die 80286-CPU benötigt eine Betriebsspannung von 5 V. Alle Ein- und Ausgänge sind TTL-kompatibel.
Zum System 80286 gehören außerdem der Taktgenerator 82284 und der Buscontroller 82288, deren Funktionen denen der entsprechenden 8086-Systemschaltkreise ähneln.
Der dazugehörige Arithmetik-Koprozessor 80287 ist code-kompatibel zum 8087. Ein Unterschied besteht jedoch in der Einbindung des Arithmetikprozessors in das System 80286. Der 80287 ist eigentlich ein Peripherieschaltkreis, das heißt, der Datentransfer mit der CPU wird über Ein- und Ausgabezyklen realisiert, wofür der E/A-Adreßbereich von 00F8H bis 00FFH reserviert ist. Für den Datentransfer werden die E/A-Buszyklen in der CPU automatisch ausgelöst.
Als Interface-Schaltkreise können im System 80286 die gleichen Typen wie im System 8086 verwendet werden, wobei aber auf die Einhaltung der Zeitbedingungen zu achten ist.
Weiterhin gibt es einige Koprozessor- und Controller-Schaltkreise, die über eine spezielle 80286-Betriebsart verfügen und so als 80286-Systemschaltkreise verwendet werden können. Dazu gehören die dRAM-Controller 8207 und 8208, der DMA-Koprozessor 82258 und der Grafik-Koprozessor 82786.

9. Multitaskverarbeitung

9.1. Grundprinzipien der Echtzeitverarbeitung

Für 8086-Rechner existieren neben Einzelnutzerbetriebssystemen (z. B. SCP 1700, CP/M 86, DCP, MS-DOS) auch Echtzeitbetriebssysteme (z. B. BOS 1810, RMX86, EMOS, RMOS2).
Echtzeitbetriebssysteme verfügen im allgemeinen über eigene Bestandteile für die Entwicklung von Programmen auf dem Assemblerniveau oder mit Hochsprachen (Editier-, Übersetzer-, Link- und Debuggprogramme). Diese benutzen eine eigene Dateiorganisation, die durch eine Vielzahl von Dienstprogrammen unterstützt wird.
Bei der Programmentwicklung mit einem Echtzeitbetriebssystem werden Subsysteme des Betriebssystems in das Anwenderprogramm übernommen, wodurch sich neue Formen der Programmorganisation ergeben. Im folgenden sollen deshalb nicht die Hilfsmittel der Programmentwicklung mit einem Echtzeitbetriebssystem, sondern die Eigenschaften des echtzeitfähigen Anwenderprogramms im Mittelpunkt der Darlegungen stehen.
Dafür werden Anwenderprogramme betrachtet, die das Subsystem **NUCLEUS** (deutsch: Kern) des Echtzeitbetriebssystems BOS 1810/1/ einschließen.
Verschiedene Möglichkeiten für die Bildung einer solchen Programmkonfiguration werden am Ende dieses Abschnitts aufgezeigt.
Die charakteristischen Eigenschaften echtzeitfähiger Programme liegen in der quasiparallelen Abarbeitung einzelner Programmteile, der **Tasks**, und der Unterstützung der **Interruptverarbeitung.**
Die Formulierung eines Anwenderprogramms aus verschiedenen Tasks für die Behandlung unterschiedlicher Ereignisse in einem Prozeß führt zu einer weitgehenden Entflechtung von Teilprogrammen und zu einer hohen Flexibilität bei der Programmentwicklung und bei Änderungen.
In einem Multi-Task-Programm wird die Abarbeitung der Tasks von einem Task-Scheduler (deutsch: Ablaufauflister) innerhalb des NUCLEUS nach dem Prioritätsprinzip gesteuert.
Tasks werden vom NUCLEUS als **Objekte** verwaltet. Weitere Objekttypen dienen der Ressourcenverwaltung des Rechners (Job), der dynamischen Speicherzuweisung (Segment), dem Informationsaustausch zwischen den Tasks (Mailbox), der Synchronisation der Taskabarbeitung (Semaphore und Region).
Jede Task erhält einen Prioritätswert, einen der Ausführungszustände: *rechenbereit* (ready), *schlafend* (asleep) oder *suspendiert* (suspended), einen eigenen Stackbereich und meist einen eigenen Datenbereich.
Die jeweils höchstpriorisierte, rechenbereite Task wird abgearbeitet. In den Wartezeiten der höchstpriorisierten Task übernimmt die nächst höher priorisierte, rechenbereite Task den Prozessor.
Wartezeiten können prozeßbedingt (z. B. bei E/A-Operationen) auftreten oder durch eine Vielzahl von Systemrufen ausgelöst werden, so daß auch niedrig priorisierte Tasks in die Abarbeitung kommen.
Der Objekttyp **Job** umfaßt ein Kontingent an Rechnerressourcen und Objekten. Unter den Tasks eines Jobs befindet sich mindestens eine Jobinitialtask. Durch die Bildung verschiedener Jobs kann der Arbeitsbereich des Rechners in voneinander abgegrenzte Bereiche aufgeteilt werden. Eine Task kann innerhalb eines Jobs dessen Ressourcen auf weitere Jobs („Kinderjobs") aufteilen.
Die Gesamtressourcen des Rechners werden von einem Wurzeljob (rootjob) verwaltet, der vom NUCLEUS erzeugt wird und dessen Initialtask (Wurzeltask) nach dem Anlauf des Systems gestartet wird.
Innerhalb der Wurzeltask werden ein oder mehrere Anwenderjobs gebildet, die die Anwendertasks enthalten.
Bild 9.1 zeigt die Struktur eines Multi-Task-Programms. Am Anfang werden in einem Initialisierungsteil die Hardwareeinstellungen (Programmierung Timer und Interruptcontroller) vorgenommen. Im NUCLEUS werden die internen Systemdaten für die Verwaltung aller Objekttypen und der Wurzeljob erzeugt.
Innerhalb eines Anwenderjobs werden Tasks kreiert, die auf die Ressourcen des Jobs zugreifen können.
Die Reaktionsfähigkeit eines Echtzeitprogramms auf externe Ereignisse wird durch die Bildung von **Interrupttasks** gewährleistet. Eine Interrupttask wird durch eine Interruptanforderung aus der Hardware in den rechenbereiten Ausführungszustand überführt.

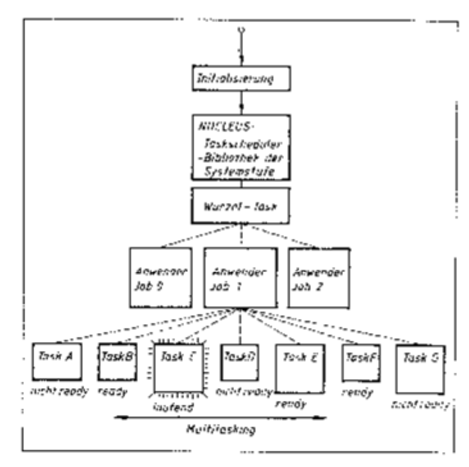


Bild 9.1 Struktur eines Multitaskprogramms

9.2. Task-, Job- und Segment-Objekte

Der NUCLEUS enthält eine Bibliothek von Systemrufen, die in Anwendertasks aufgerufen werden können. Die Maximalversion des NUCLEUS von 24 KByte verfügt über 56 Systemrufe. Kleinere NUCLEUS-Versionen mit einem Subset von Systemrufen lassen sich im Konfigurationsprozeß festlegen. Alle Objekte werden durch Systemrufe erzeugt und gelöscht.

9.2.1. Erzeugen von Tasks

Nach der Vorgabe einer Priorität (Wert 0 - 255; 0: höchste Priorität), der Startadresse des Task-Programms, eines Wertes für das

Datensegment, des Taskstackpointers, der Stackgröße, des Taskflags für die Angabe der 8087-Koprozessornutzung und eines Pointers für die Ablage des Fehlercodes im Stack der aufrufenden Task, erzeugt der Systemruf **CREATE_TASK** eine Task für das angegebene Programm. Dabei entstehen die Task-Systemdaten, das heißt ein 80 Byte großer Steuerblock mit den Task-Informationen.
Wie jedes Objekt wird eine Task durch einen **Token,** einen 16-Bit-Idendifikator, gekennzeichnet. Der Token enthält zugleich die Adreßinformation der Task-Systemdaten.
Eine neu erzeugte Task besitzt den Ausführungszustand *rechenbereit.*
Der Task-Scheduler erzeugt eine Verkettung aller rechenbereiten Tasks durch die Aufnahme der Token der nächst höher und nächst niedriger priorisierten Tasks in die Systemdaten einer jeden Task.
Eine zweite gelinkte Liste von Task-Systemdaten existiert für die nicht rechenbereiten Tasks. Beim Bilden und Löschen von Tasks oder bei der Änderung des Ausführungszustandes von Tasks werden die Listen neu geordnet.
Der Systemruf **DELETE_TASK** mit der Vorgabe des Task-Tokens im Stack löscht die Task.

9.2.2. Erzeugen von Jobs
Der Objekttyp *Job* verwaltet einen Speicherbereich (memory pool) und eine Menge von Objekten.
Ein Job enthält einen Katalog (Jobverzeichnis, job directory), in dem für Objekte die Zuordnung von Namen zu den Objekttoken eingetragen werden kann. Da für den Zugriff auf jedes Objekt der Token benötigt wird, bietet der Katalog die Möglichkeit, die Token von in anderen Tasks erzeugten Objekten zu bestimmen. Bei der Bildung eines Jobs mit dem Systemruf **CREATE_JOB** werden u.a. die Größe des Jobverzeichnisses, der Speicherbedarf, die Maximalzahl der Objekte, die Maximalzahl der Tasks und alle Parameter für die Initialtask vorgegeben. Ein erzeugter Job wird durch einen Jobtoken gekennzeichnet.
Der NUCLEUS vergibt den von den Objekten beanspruchten Speicher mit fallender Adresse.
Nach Bild 9.2 wird bei der Erzeugung eines Jobs zuerst das Jobobjektverzeichnis an der Obergrenze des noch verfügbaren Arbeitsspeichers angelegt. Daran schließen sich die Jobsystemdaten an.
Für die mit dem Job erzeugte Initialtask wird zunächst der zugeordnete Stackbereich reserviert. Darunter liegen die Task-Systemdaten.
Der Systemruf **DELETE_JOB** löscht einen Job.

9.2.3. Erzeugen von Segmenten
Der innerhalb des umgebenden Jobs verfügbare Arbeitsspeicher kann von einer Task portionsweise angefordert werden. Der Systemruf **CREATE_SEGMENT** mit der Vorgabe der Speichergröße erzeugt ein Segmentobjekt. Segmente enthalten z.B. den Programmcode für die Tasks (Bild 9.2.). Der zugewiesene Speicherbereich wird mit dem Systemruf **DELETE_SEGMENT** wieder freigegeben.

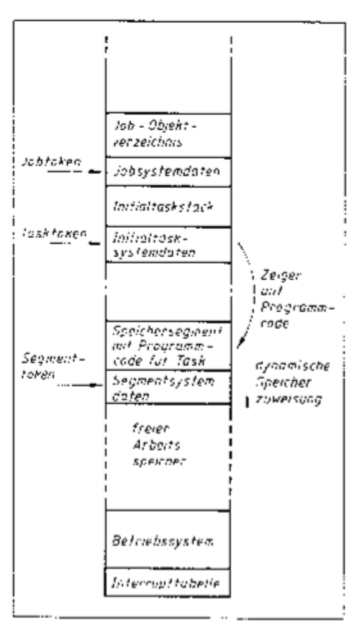


Bild 9.2 dynamische Speicherzuweisung bei der Bildung von Objekten

9.2.4. Taskumschaltung
Das dem BOS 1810-NUCLEUS zugrunde liegende Prioritätsprinzip setzt voraus, daß eine in der Abarbeitung befindliche Task selbständig den Prozessor, zumindest zeitweise, freigibt.
Zu den einfachsten Möglichkeiten zählen die Systemrufe für die zeitweilige Prozessorfreigabe durch die aktive Task.
Der Systemruf **SLEEP** versetzt die angegebene Task (meist die aktive) für eine vorgegebene Zeit in den Abarbeitungszustand *schlafend.* Nach dem Ablauf dieser Zeit erhält die Task automatisch wieder den Abarbeitungszustand *rechenbereit.* Der Systemruf **SUSPEND_TASK** suspendiert die aktive oder eine andere Task von der Abarbeitung. Durch mehrfaches Suspendieren wird eine entsprechende Suspendierungstiefe eingestellt.
Eine suspendierte Task kann von der aktiven Task mit dem **RESUME_TASK**-Ruf wieder in einen *rechenbereiten* Abarbeitungszustand gebracht werden. Für eine mehrfach suspendierte Task sind mehrere RESUME_TASK-Rufe notwendig.

9.2.5. Systemrufe für den Objektzugriff
Für Informationen über die Objekte steht eine Anzahl von Systemrufen zur Verfügung.
Über den Systemruf **GET_TASK_TOKENS** können die Token der laufenden Task, des sie enthaltenden Jobs oder des Wurzeljobs bestimmt werden.
Die Priorität einer Task wird mit dem Systemruf **GET_PRIORITY** ermittelt.
Der Token eines beliebigen Objektes kann unter einem bis zu 12stelligen Namen in das Verzeichnis eines Jobs mit dem Ruf **CATALOG_OBJECT** ein- und mit dem Ruf **UNCATALOG_OBJECT** ausgetragen werden.
Für katalogisierte Objekte ergibt der Ruf **LOOKUP_OBJECT** bei Vorgabe des Namens den Token.

9.3. Task-Kommunikation und -Synchronisation
9.3.1. Informationsaustausch zwischen den Tasks

Die weitgehende Unabhängigkeit in der quasiparallelen Abarbeitung der einzelnen Tasks setzt spezielle Techniken für die Informationsübermittlung zwischen den Tasks voraus.
Diesem Zweck dienen Mailboxobjekte. Eine Mailbox besteht nach Bild 9.3 aus zwei Warteschlangen. In der Objektwarteschlange können Nachrichten abgespeichert werden. Für eine Nachricht steht in der Objektwarteschlange der Token eines beliebig großen Segmentes mit den zu übertragenden Informationen. Die Segmenttoken können von allen Tasks, die eine Zugriffsmöglichkeit auf die Mailbox besitzen, eingeschrieben werden. Für den Zugriff auf eine Mailbox benötigt eine Task lediglich deren Token.

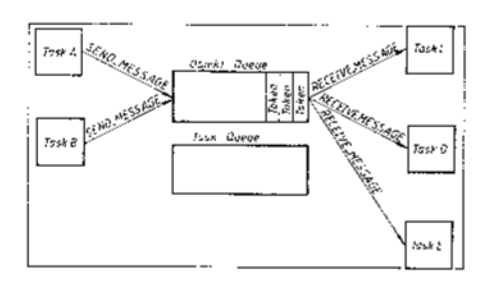


Bild 9.3 Objekttyp Mailbox

Mit der Kenntnis des Mailboxtokens kann eine Task von einer Mailbox auch Nachrichten anfordern. Falls mehr Nachrichtenanforderungen als Segmenttoken für eine Mailbox vorliegen, entsteht eine Warteschlange von Tasks.
Eine Mailbox wird von einer Task mit dem **CREATE_MAILBOX**-Ruf erzeugt. Dabei kann die Organisationsform und Größe der Warteschlangen angegeben werden. Die Bekanntgabe des Tokens für andere Tasks kann z.B. durch Katalogisieren im Objektverzeichnis eines übergeordneten Jobs erfolgen.
Der Token des Segmentes, das die Information enthält, kann mit dem **SEND MESSAGE**-Systemruf in eine Mailbox geschrieben werden.
Mit dem Ruf **RECEIVE_MESSAGE** wird eine Nachricht von der Mailbox gelesen. Eine Leseanforderung an eine Mailbox, die keine Nachricht enthält, versetzt die anfordernde Task in den Abarbeitungszustand *schlafend.* In diesem Fall gibt die anfordernde Task den Prozessor frei, bis in der Mailbox eine Nachricht eingegangen ist. Dadurch kommt die anfordernde Task wieder in den *rechenbereiten* Abarbeitungszustand.
Der Systemruf **DELETE_MAILBOX** löscht eine Mailbox.

9.3.2. Taskumschaltung über Semaphore
Der Objekttyp **Semaphore** (Zeichenträger) dient der Synchronisation der Taskabarbeitung. Durch die Anforderung von *Einheiten* (units) des Kontingentes eines Semaphores mit dem Systemruf **RECEIVE_UNITS** kann die

Abarbeitung einer Task bedingt unterbrochen werden. Falls die Anzahl der angeforderten Einheiten nicht in dem Semaphore enthalten ist, geht die anfordernde Task in den Zustand *schlafend*, und der Token der anfordernden Task wird in die Taskqueue eingetragen. Nach der Zuführung von Einheiten an das Semaphore mit dem Systemruf **SEND_UNITS** durch eine aktive Task wird die an der Mailbox wartende Task wieder *rechenbereit*.
Nach Bild 9.4 enthält ein Semaphore eine Taskqueue, in der alle anfordernden Tasks nach einem einstellbaren Organisationsprinzip (FIFO oder Prioritätsordnung) abgespeichert werden, falls das Kontingent an Einheiten für die erste anfordernde Task nicht ausreicht.
Semaphore können mit den Systemrufen **CREATE_SEMAPHORE** und **DELETE_SEMAPHORE** erzeugt und gelöscht werden.

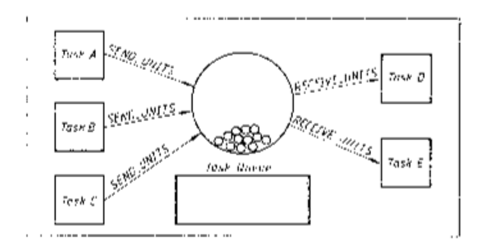


Bild 9.4 Objekttyp Semaphore

9.3.3. Datenzugriffssteuerung mit Regionen
Der ungestörte Zugriff auf Daten (oder die Abarbeitung von Teilprogrammen) kann durch Objekte vom Typ **Region** geschützt werden.
Der Zugriffsschutz wird durch eine Task mit dem Systemruf **RECEIVE_CONTROL** an die Region angefordert. Eine höherpriorisierte Task, die ebenfalls den **RECEIVE_CONTROL**-Systemruf an die Region enthält, kann auf die gemeinsamen Daten erst zugreifen, wenn diese von der vorherigen Task vervollständigt worden sind.
Da bereits die Anforderung der vorherigen Task in der Taskqueue der Region eingetragen ist, geht die höherpriorisierte Task in den Zustand *schlafend*, und die vorherige Task setzt die Abarbeitung fort. Die nun wartende Task wird in die Taskqueue der Region eingetragen.
Der Systemruf **SEND_CONTROL** löscht die Zugriffsanforderung der aktiven Task in der Region.
Eine alternative Form der Zugriffsanforderung bietet ein weiterer Systemruf **ACCEPT_CONTROL**, die nur bedient wird, wenn keine früheren Anforderungen in der Taskqueue der Region vorliegen.
Regionobjekte werden mit den Systemrufen **CREATE_REGION** und **DELETE_REGION** erzeugt und gelöscht.

9.4. Echtzeitverarbeitung

Die schnelle Reaktion auf externe Ereignisse ist eine der Hauptaufgaben eines Echtzeitbetriebssystems. Dafür wird die im System 8086 enthaltene Interruptorganisation mit der Multitaskverarbeitung kombiniert.
Nach der im Abschnitt 4 beschriebenen Interruptorganisation muß die Bearbeitung für ein externes Ereignis durch ein Anforderungssignal an einem INTR-Eingang eines Slave- oder des Master-PIC (programmable interrupt controller) angefordert werden.
Die in der Hardware ausgewählte Anforderung mit der höchsten Interruptpriorität führt im Fall der Interruptfreigabe zur Unterbrechung jeder beliebig hoch priorisierten Task. Mit der eingeschobenen Interrupt-Service-Routine können die für das externe Ereignis benötigten Reaktionen sofort ausgelöst werden. Mit dieser Methode, die für die Bearbeitung dringender Fälle auch möglich ist, wird jedoch die Prioritätsorganisation der Multitaskverarbeitung übergangen.
Im allgemeinen Fall soll auch die Bearbeitung externer Ereignisse der Prioritätsentscheidung der Multitaskverarbeitung untergeordnet werden. Dafür werden **Interrupttasks** eingesetzt.
Eine Interrupttask wird wie jede Task mit einem CREATE_TASK-Systemruf erzeugt und durch ein zusätzliches Interruptprogramm, den Interrupthandler, in der Abarbeitung gesteuert. Durch den Systemruf **SET_INTERRUPT** innerhalb der Interrupttask wird diese einem INTR-Eingang eines Slave- oder des Master-PIC zugeordnet. Zugleich wird die Adresse des zugehörigen Interrupthandlers in die Interrupttabelle eingetragen. Der Systemruf bildet außerdem einen Puffer, in dem die Anforderungen für den entsprechenden INTR-Eingang abgespeichert werden.
Nach Bild 9.5 wird eine Interrupttask nach ihrer Bildung entsprechend der Taskpriorität gestartet. Nach der Interrupthandlerzuordnung versetzt der Systemruf **WAIT_INTERRUPT** die Interrupttask in den Zustand *schlafend*.
Eine durch ein externes Ereignis ausgelöste und an die CPU durchgeschaltete Interruptanforderung führt zur sofortigen Ausführung des zugehörigen Interrupthandlers. Diese enthält im wesentlichen nur den Systemruf **SIGNAL_INTERRUPT**, durch den die Interrupttask *rechenbereit* wird.

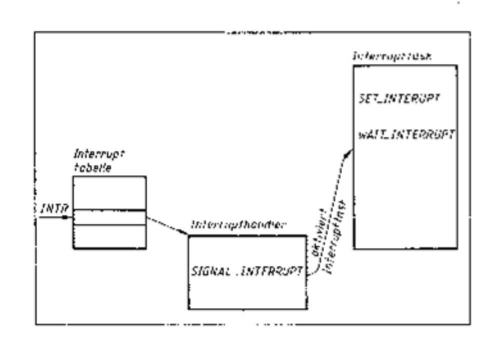


Bild 9.5 Interruptbehandlung mit Interrupttask

9.5. E/A-Operationen

In einem Echtzeitprogramm muß der Nutzung der bei E/A-Operationen auftretenden Wartezeiten besondere Aufmerksamkeit gewidmet werden.
Für E/A-Operationen existieren im Echtzeitbetriebssystem BOS 1810 die Subsysteme BIOS (basic I/O system) und EIOS (extended I/O system).
Für spezielle Zielrechnerkonfigurationen ist es oft günstiger, anwendungsspezifische E/A-Systeme in Anlehnung an die Organisationsprinzipien des BIOS zu entwickeln. Das Subsystem BIOS verwendet im Prinzip die im folgenden beschriebene Organisationsform.
Einheitlich für alle Geräte werden asynchrone Systemrufe für das Lesen (**READ**) und Schreiben (**WRITE**) von Daten benutzt. Der Aufruf einer E/A-Operation erzeugt einen E/A-Anforderungsblock (I/O request segment, IORS) mit allen E/A-Zugriffsinformationen. Die Anforderung wird in eine Warteschlange von Anforderungsblöcken (IORS queue) eingeordnet. Die Warteschlange entsteht nach Bild 9.6 durch vor- und rückwärts gerichtete Zeiger in jedem Block auf die benachbarten Blöcke. Nach dem Aufruf einer E/A-Operation kann die aufrufende Task die Abarbeitung sofort fortsetzen. Im allgemeinen Fall geht die aufrufende Task vor der Verwendung der angeforderten Daten mit einem RECEIVE_MESSAGE-Systemruf in den Zustand *schlafend* und gibt den Prozessor frei. Der Token der Mailbox, die die Verfügbarkeit der Daten nach der Datenübertragung meldet, wird beim E/A-Aufruf als Parameter angegeben.
Für jedes E/A-Gerät existiert je eine Warteschlange von E/A-Anforderungsblöcken.
Jede Warteschlange wird von einer hochpriorisierten Queuetask bedient. Die Queuetask übergibt bei Bereitschaft des E/A-Gerätes die Zugriffsparameter an den Gerätetreiber und gibt danach den Prozessor frei.
Das E/A-Gerät meldet die Übertragungsbereitschaft mit einem Interrupt. Dieser aktiviert eine Interrupttask, die die Datenübertragung zwischen dem Gerät und einem Pufferspeicher ausführt.
Die Interrupttask meldet den Abschluß der Datenübertragung an die Queuetask, die die Fertigmeldung an die Mailbox in der aufrufenden Task weitergibt und gegebenenfalls eine weitere E/A-Anforderung an den Gerätetreiber übergibt.
Die aufrufende Task wird durch die Fertigmeldung an die Mailbox wieder *rechenbereit*.
Nach diesem Prinzip können Anforderungen an alle E/A-Geräte asynchron von jeder Task und aus jedem Job gestellt werden. Die E/A-Subsysteme des BOS 1810 bieten zusätzlich Möglichkeiten der Vergabe von Zugriffsberechtigungen zu E/A-Geräten an die Jobs.

9.6. Echtzeitprogrammentwicklung

Die Entwicklung eines Echtzeitprogramms erfolgt im allgemeinen Fall mit der Entwicklungstechnik eines Echtzeitbetriebssystems. Im Echtzeitbetriebssystem BOS 1810 können die Systemrufe der Subsysteme in Assembler- und PL/M-86-Programmen benutzt werden. Mit Hilfe des Link-Programms wird eine Interfacebibliothek für den Anschluß der Systemrufe in das 8086-Maschinenprogramm eingebunden.
Echtzeitprogramme können jedoch auch auf anderen Betriebssystemen (SCP 1700, MS-DOS) auf dem Assemblerniveau oder in der Programmiersprache C entwickelt werden.

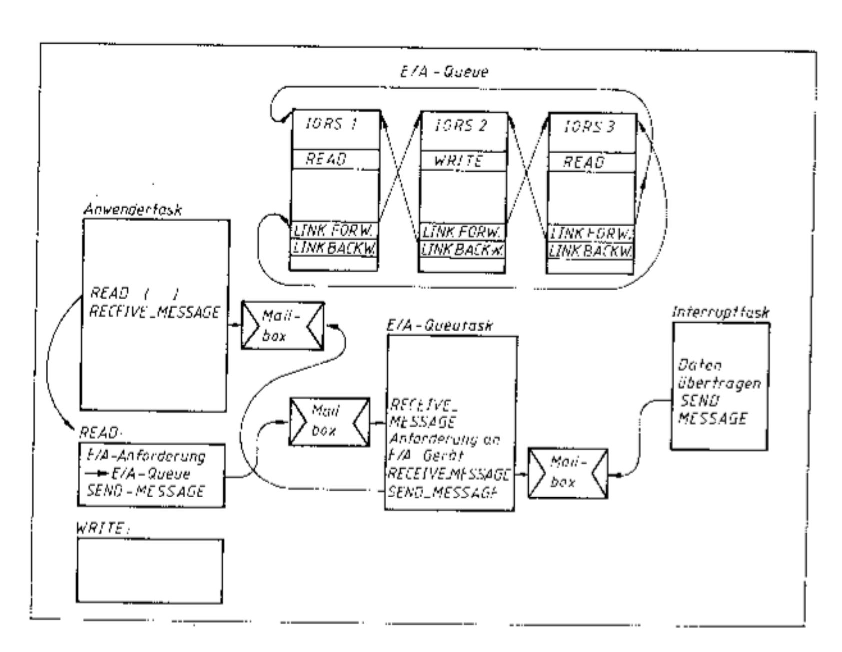


Bild 9.6 Organisation der taskgesteuerten E/A-Operationen

Eine Multitaskergänzung des Wirtsbetriebssystems steuert die Multitaskverarbeitung. Ein als Task laufendes spezielles Ladeprogramm (Applikationslader) für die Dateiformen der ausführbaren Programme (z. B. .CMD-Dateien) erzeugt die Tasks.

Dabei können die Anwenderprogramme auch die Systemrufe des Wirtsbetriebssystems nutzen.

9.7. Systemdebugger

Die Testung eines Echtzeitprogrammes erfordert spezielle Hilfsmittel für die Diagnose der Objekteigenschaften. Der Systemdebugger des Betriebssystems BOS 1810 enthält unter anderem die folgenden Kommandos für die Diagnose der im Echtzeitprogramm enthaltenen Objekte:

vj [<jobtoken>]
Darstellung der Jobhierarchie vom angegebenen Job an

vk
Angabe der Listen der *rechenbereiten* und *nicht rechenbereiten* Tasks

vo <jobtoken>
Angabe aller Objekte des Jobs

vd <jobtoken>
Angabe des Objektverzeichnisses des Jobs

vt
Angabe der Objektattribute

Auch ohne einen Systemdebugger können die Objektattribute aus der Analyse der Objekt-Systemdaten gewonnen werden.

9.8. Programmbeispiel

Bild 9.7 enthält einen Assemblerprogrammausschnitt eines Multitaskprogramms. Ein in der Assemblerprogrammliste enthaltenes Teilprogramm TSK1 wird mit dem Systemruf CREATE_TASK als Task kreiert. Der Anschluß des Systemrufes an das Subsystem NUCLEUS wird mit dem Unterprogramm NCRTSK vollzogen, das das zugehörige Interfacebibliotheksprogramm darstellen soll.

Wenn das laufende Programm den Prozessor freigibt, wird die erzeugte Task, falls keinen anderen höherpriorisierten Tasks *rechenbereit* sind, zur Ausführung gelangen, ohne daß ein CALL-Befehl ausgeführt wurde.

9.9. Objektorientierte Programmiertechniken

Die objektorientierte Programmierung findet in zunehmendem Maße in modernen höheren Programmiersprachen und für die Organisation der Parallelverarbeitung Anwendung. Die Erweiterungen beziehen sich vor allem auf die Definition einer Vielzahl anwenderspezifischer Objekttypen.

Bei der quasiparallelen oder echt parallelen Abarbeitung der Objekte (Architekturen mit mehreren Prozessoren) werden Programme und Daten von den übrigen Programmteilen vollständig isoliert. Informationen zwischen den Objekten werden nur über Nachrichten ausgetauscht.

Neben diesen Vorteilen kann die objektorientierte Programmierung in vielen Fällen für eine vereinfachte implizite Programmformulierung genutzt werden.

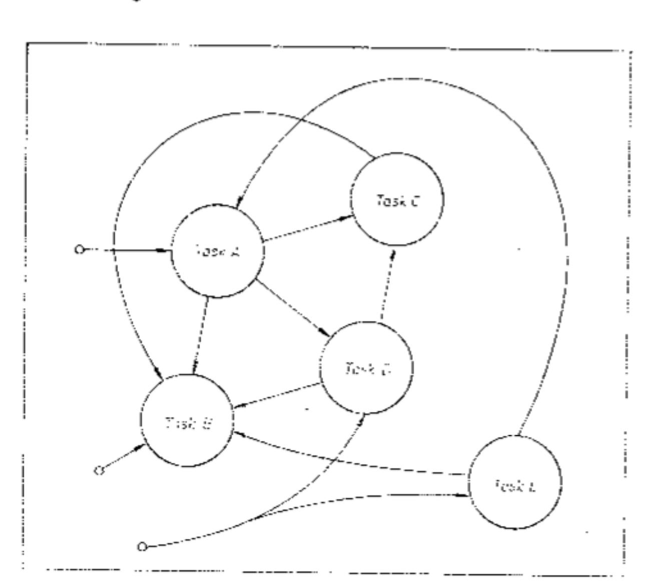
Im folgenden sollen diese Eigenschaften bei der Anwendung der NUCLEUS-Objekte demonstriert werden. Die dabei entstehenden schlechteren Abarbeitungseigenschaften

```
DATA    DSEG
TSKTOK  RW      1
PRTY    DW      200
STSIZE  DW      400H
ECODE   RW      1
        .
        .
CODE    CSEG
        .
        .
        .                           ;Vorgabeparameter für Systemruf
                                    ;CREATE_TASK in Stack schreiben:
        PUSH WORD PTR PRTY          ;Taskpriorität
        PUSH CS                     ;Programmadresse, Segmentanteil
        MOV  AX,OFFSET TSK1
        PUSH AX                     ;Programmadresse, Offsetanteil
        MOV  AX,0
        PUSH AX                     ;Wert für DS
        PUSH AX                     ;Stackpointerzuweisung durch
        PUSH AX                     ;NUCLEUS
        PUSH WORD PTR STSIZE        ;Stackgrösze
        PUSH AX                     ;Taskflag (keine 8087-Nutzung)
        PUSH DS                     ;Zeiger für Fehlercode
        MOV  AX,OFFSET ECODE
        PUSH AX
        CALL NCRTSK                 ;Interfacebibliotheksprogramm
        MOV  TSKTOK,AX              ;Tasktoken abspeichern
        .
        .                           ;Vorgabeparameter für Systemruf
                                    ;DELETE_TASK in Stack schreiben:
        MOV  AX,0
        PUSH AX                     ;laufende Task löschen
        PUSH DS                     ;Zeiger für Fehlercode
        MOV  AX,OFFSET ECODE
        PUSH AX
        CALL NDLTSK                 ;Interfacebibliotheksprogramm
TSK1:   .                           ;Assemblerprogramm für zu
        .                           ;erzeugende Task
        JMP  TSK1

                                    ;Eintrittspunkte in die Inter-
                                    ;facebibliothek für NUCLEUS-
                                    ;Systemcalls:
NCRTSK: .                           ;CREATE_TASK
        .
NDLTSK: .                           ;DELETE_TASK
        .
```

Bild 9.7 Assemblerprogrammausschnitt mit der Bildung einer Task

Bild 9.8 Objektorientierte Programmiertechnik am Beispiel der Entflechtung eines Netzes



(größere Verarbeitungszeit) im Vergleich zu konventionellen Lösungen sind auch für die objektorientierte Programmierung typisch.
Problemstellungen mit Signalflußcharakter nach Bild 9.8 können in die Bearbeitung von Teilproblemen zergliedert werden. Dabei ist es notwendig, daß für die Bearbeitung bestimmter Teilsysteme die Ergebniswerte anderer Teilsysteme vorliegen müssen. In konventionellen Programmlösungen muß deshalb zuerst die Berechnungsreihenfolge der Teilsysteme festgelegt werden.
In der objektorientierten Programmierung wird die Berechnung aller Teilsysteme quasiparallel mit je einer Task gestartet. Die Startreihenfolge ergibt sich aus den willkürlich zugeordneten Taskprioritäten. Alle für die Verkopplung der Teilsysteme benötigten Informationen werden mit NUCLEUS-Systemrufen über Mailboxen vermittelt.
Die Bearbeitung eines Teilproblems wird automatisch unterbrochen, bis dafür benötigte Ergebnisse von anderen Objekten verfügbar sind. Damit wird der Prozessor für die Bearbeitung anderer Teilsysteme freigegeben. Auf diese Weise stellt sich auf der Grundlage der Multitasksteuerung die notwendige Reihenfolge der Bearbeitung der Teilsysteme ein. Dieser Mechanismus kann auch für die Behandlung einer Dead-lock-Situation, in der mehrere Tasks gegenseitig auf Nachrichten warten, genutzt werden. In diesem Fall wird eine niedrig priorisierte Task aktiv, die diese Situation bedient.

10. Programmentwicklung in C

In diesem Kapitel wird der C-Compiler *DRC* vorgestellt. Ziel des Beitrages ist es, Hinweise für die Bedienung und Nutzung des C-Compilers DRC und die Implementation der Hochsprache **C** auf dem K 1810 WM86 (8086) unter dem Betriebssystem CP/M-86 zu geben. Dabei wird auf den bekannten Veröffentlichungen /1, 2, 3/ zur Anwendung der Programmiersprache C aufgebaut.

10.1. Systemprogramme für die Programmierung in C

Das Herangehen bei der Entwicklung von C-Programmen entspricht der für Assemblerprogrammierung in Abschnitt 6 beschriebenen Weise. Zunächst werden die Quellprogramme editiert und dann mit Hilfe des C-Compilers in Objektdateien *.obj übersetzt.
Nach dem Linken eines bzw. mehrerer Programme zu einem CMD-File wird das Programm durch Aufrufen des Namens gestartet.
Die Fehlersuche erfolgt günstig auf Maschinenniveau mit dem symbolischen Debugger *SID86* (unter Zuhilfenahme eines Reassemblerlistings) oder durch zweckmäßiges Einfügen von Ausgabeanweisungen in das Programm (z. B. Anzeige lokaler Variablen), die bei nachgewiesener Fehlerfreiheit des Programms wieder entfernt werden können.

10.1.1. Bestandteile und Arbeitsweise des C-Compilers DRC

Das gesamte C-Compilerprogrammpaket setzt sich aus den folgenden Teilprogrammen zusammen:

DRC.CMD	– Basismodul des C-Compilers
DRC860.CMD	– Präprozessor
DRC861.CMD	– Codegenerator
R.CMD	– Programm zum Nachladen des Codegenerators
DRC862.CMD	– Disassembler
DRC.ERR	– Fehlerbeschreibungsdatei
DRCRPP.CMD	– Rückübersetzer für Präprozessor

Zusätzlich sind die Laufzeitbibliotheken **CLEARS.L86** und **CLEARL.L86** sowie für bestimmte Anwendungen die Header-Dateien **STD.H, PORTAB.H, CTYPE.H** und **VAR.H** erforderlich.

Die Übersetzung eines C-Programms geht wie folgt vor sich: Zunächst wird das Quellprogramm vom Präprozessor vorübersetzt. Dabei werden die Präprozessoranweisungen wie Dateieinfügungen, Makrosubstitutionen und bedingte Generierungsanweisungen ausgeführt. Die entstehende Datei befindet sich auf dem aktuellen Laufwerk und hat den Namen CTEMP.TOK. Die für DRC860 möglichen Präprozessoranweisungen lauten:

#define
Makrodefinitionen mit und ohne Parameter. Eine Makrodefinition kann maximal 10 Parameter enthalten.
#undef
löscht eine Makrodefinition
#include<d:file>
Einfügen der Datei *file* vom angegebenen Laufwerk. Wurde kein Laufwerk angegeben, wird die Datei vom durch die –i-Option (siehe 8.1.3.) festgelegten Laufwerk geladen.
#include"d:file"
Einfügen der Datei *file* vom angegebenen Laufwerk. Wurde kein Laufwerk angegeben, wird das aktuelle Laufwerk benutzt.
#if, #ifdef, #ifndef, # endif, #else
bedingte Compilierung des folgenden Quelltextes
#nolist
Diese im C-Standard /1, 2/ nicht vorkommende Präprozessoranweisung bezieht sich auf die Generierung des Disassemblerlistings durch DRC862. Der nachfolgende Quelltext wird bei der Auflistung unterdrückt.
#list
Der nachfolgende Text wird in das Disassemblerlisting aufgenommen.

Die Präprozessoranweisung *#line* ist im DRC860 *nicht* implementiert.
Die Datei CTEMP.TOK kann mit Hilfe des Programms DRCRPP in eine lesbare Darstellung überführt werden, so daß die Richtigkeit des Textersatzes geprüft werden kann.
Der C-Codegenerator verarbeitet die Datei CTEMP.TOK und erzeugt die Objektdatei. Mit Hilfe des Disassemblers DRC862.CMD ist es möglich, ein Listing zu schaffen, das die C-Anweisungen und deren Umsetzung in Assemblersprache des 8086 enthält. Treten während der Verarbeitung des Quellprogramms Fehler auf, werden diese mit Nummern spezifiziert. Mit Hilfe der Datei DRC.ERR erfolgt eine Interpretation der Fehlerursache.

Die verschiedenen Subprogramme werden vom Programm DRC.CMD aufgerufen. Dabei können verschiedene Optionen für die Spezifizierung der Abarbeitung und das zu erzeugende Programm gesetzt werden.

10.1.2. Aufruf des C-Compilers

Zum Start des C-Compilers wird dieser mit dem Namen DRC aufgerufen:

DRC [d:]cprog [–option . . .]

Dabei ist *cprog* der Name der zu übersetzenden Quelldatei. Wird die Datei *cprog* nicht gefunden, wird die Datei *cprog.c* auf dem spezifizierten Laufwerk gesucht. Mit Hilfe von Optionen kann der Übersetzungsvorgang gesteuert und die Abarbeitung der Subprogramme beeinflußt werden.

Beispiel:
A>DRC TEST.C

Der Compiler sucht die Datei TEST.C auf Laufwerk A, und wenn sie vorhanden ist, wird mit der Übersetzung begonnen.

Beispiel:
A>DRC B:TEST

Jetzt sucht der C-Compiler die Datei TEST auf Laufwerk B. Falls er sie nicht findet, sucht er anschließend die Datei TEST.C. Falls keine der beiden Dateien vorhanden ist, folgt eine Fehlerausschrift.
Durch Betätigen einer beliebigen Taste kann die Arbeit des C-Compilers unterbrochen werden. Es erfolgt dann die Ausschrift

STOP DRC (Y/N)?

und mit der entsprechenden Eingabe wird der Compilerlauf abgebrochen.

10.1.3. Optionen des C-Compilers

Zusätzlich zur Angabe des Quellennamens können verschiedene Optionen, die die Arbeitsweise des C-Compilers beeinflussen, angegeben werden. Diese Optionen können in beliebiger Reihenfolge auftreten und beginnen stets mit einem "–". Bei Angabe eines wahlweisen Parameters ist darauf zu achten, daß zwischen Option und folgendem String kein Leerzeichen stehen darf.
Einige der wichtigsten Optionen sollen hier in alphabetischer Reihenfolge vorgestellt und erläutert werden. Anschließend werden dann einige Beispiele für die Anwendung und die Wirkung von Optionen gezeigt.
–a[string]
ist eine Steueroption für das Programm DRC.CMD und bewirkt das automatische Linken des geschaffenen Programms im Anschluß an die Übersetzung mit Hilfe des Linkers LINK86. Dabei ist *string* die Parameterzeile für LINK86 gemäß 6.4.2. Wird kein *string* angegeben, wird nur das vom C-Compiler in Bearbeitung befindliche Programm gebunden.
–b
Auswahl des Übersetzungsmodells
Bei Setzen dieser Option wird ein Programm für das Large-Modell geschaffen.
–f
bewirkt die Generierung von 8087-Befehlen und ermöglicht so die Einbeziehung des Arithmetikprozessors.

–id:
spezifiziert das Laufwerk **d:**, von dem die Datei <name> der #include-Anweisung des Präprozessors geladen wird.
–L[d:]name
Schaffung eines Listings, das die Include-Dateien auflistet und die Verschachtelungstiefe von Blöcken anzeigt. Wird kein *name* angegeben, erfolgt die Ausgabe der Listdatei auf dem Bildschirm.
–o[d:]name
spezifiziert das Laufwerk und den Namen des vom Codegenerator geschaffenen Objektfiles. Ist diese Option nicht gesetzt, wird der Name des Quellfiles mit dem Nachsatz .OBJ versehen.
–p
beschränkt die Abarbeitung auf den Präprozessordurchlauf.
–r[d:][name]
Es wird der Reassembler DRC862.CMD aufgerufen, und auf Laufwerk *d:* wird die Datei *name* generiert, die das C-Programm und die entsprechenden Maschinenbefehle enthält. Ist kein Dateiname angegeben, erfolgt die Ausgabe auf dem Bildschirm.
–snumber
beschränkt die Zahl der signifikanten Zeichen jedes Symbols auf *number.* Wird die Option *–s* nicht gesetzt, sind die ersten 40 Zeichen eines Symbols signifikant. Fehlt die Angabe *number,* entstehen Fehler beim Präprozessorlauf, da dann Symbollänge Null angenommen wird und die verschiedenen Symbole nicht unterschieden werden können.
–0d:
der Präprozessor wird nicht vom aktuellen Laufwerk, sondern vom Laufwerk **d:** aufgerufen.
–1d:
R.CMD und *DRC861.CMD* werden von Laufwerk **d:** aufgerufen.
–2d:
Starten des Reassemblers *DRC862* von Laufwerk **d:**
–3d:
Starten des Linkers *LINK86* von Laufwerk **d:** (nur bei gesetzter –a-Option).
–4d:
Laden der Fehlerinterpretationsdatei *DRC.ERR* von Laufwerk **d:**
–?
Diese Option zeigt während des Compilerlaufes die gesetzten Optionen an.

Beispiele:
A>DRC test –s8

Von allen Symbolen werden nur die ersten acht Zeichen ausgewertet. Damit ist der übliche Standard eingestellt.

A>DRC test –a –b

Dieses Kommando wird als
A>DRC test
A>LINK86 test
interpretiert. Die Codegenerierung erfolgt für das Large-Modell.

A>DRC e:test –ab:prog=test1, e: test, test2

Nun kommt die Folge
A>DRC e:test
A>LINK86 b:prog=test1 e:test, test2
zur Ausführung.

E>A:DRC test –ia: –a –0a: –1a: –3b: –4a:

Es wird die auf dem aktuellen Laufwerk E befindliche Datei *test* (bzw. *test.c*) übersetzt. Dabei werden Präprozessor, der Codegenerator und die Fehlerdatei auf Laufwerk *A* gesucht. Nach der Generierung des Objektfiles wird auf dem Laufwerk *B* das Programm LINK86 gestartet. Man erhält ein Programm *test.cmd,* das durch Aufruf von *test* ausgeführt werden kann. Objektdatei, CMD-Datei und alle temporären Dateien werden auf Laufwerk *E* angelegt.

A>DRC test –re:test.lst

Alle benötigten Programme befinden sich diesmal auf dem Laufwerk *A.* Als Ergebnis dieses Programmaufrufs entsteht auf dem Laufwerk *E* ein Reassemblerlisting mit dem Namen TEST.LST.

10.1.4. Linken des Programms
Nachdem alle Quellmodule eines Programms übersetzt wurden, müssen sie gebunden (gelinkt) werden, um eine ausführbare Programmdatei zu erhalten. Das ist aus verschiedenen Gründen notwendig. Erstens sind die vom Codegenerator erzeugten Objektdateien nicht ausführbar, und zweitens benutzen die meisten Programme Funktionen, die nicht in der gleichen Datei vereinbart wurden. Solche externen Funktionen können durch den Nutzer in anderen Programmen vereinbart werden und separat zu den Objektdateien übersetzt worden sein oder sich in Programmbibliotheken befinden. Diese Bibliotheken, die die auf den meisten Systemen vorhandenen Standardfunktionen (z. B. *printf(), scanf(), fgetc()...)* enthalten, sind im Falle des DRC-Compilers die Dateien CLEARS.L86 und CLEARL.L86. Je nach dem bei der Übersetzung gewählten Speichermodell werden die benötigten Funktionen vom Linker selbständig in der entsprechenden Bibliothek gesucht und in die ausführbare Programmdatei eingebunden. Die Bibliotheksdatei muß sich dabei auf dem aktuellen Laufwerk befinden.
Obwohl die Programmiersprache **C** eine Funktion *main()* als Startpunkt eines C-Programms definiert, sind eine Reihe von systemabhängigen Programmen notwendig, bevor *main()* aufgerufen wird. Solche Programme fragen beispielsweise die Versionsnummer des Betriebssystems und die Verfügbarkeit eines Arithmetikprozessors ab. Außerdem stellen sie die Argumente *argc* und *argv* für das Hauptprogramm bereit. Auch diese Funktionen sind in den Bibliotheken enthalten. Bei den Argumenten der Hauptfunktion gibt es noch eine Besonderheit: Normalerweise zeigt der Pointer **argv[0]* auf den Namen des laufenden Programms. Da dieser Name unter CP/M-86 zur Laufzeit aber nicht verfügbar ist, enthält *argv[0]* einen Zeiger auf den Namen *Cprogram.*
Das Binden der Programme wird durch den LINK86 realisiert, der entweder im Anschluß an den Compilerlauf separat gestartet wird oder bereits vom C-Compiler aufgerufen wird (–a-Option). Dabei müssen sich die Bibliotheksdateien auf dem aktuellen Laufwerk befinden. Sie brauchen jedoch nicht in der Kommandozeile für den Linker mit angegeben werden, sondern die entsprechenden Module werden automatisch eingebunden.

10.1.5. Symbolisches Debugging
Die Debugger *DDT86* und *SID86* wurden bereits in Abschnitt 6.5. ausführlich beschrieben (siehe auch /4/). Für die Testung und Einzelschrittabarbeitung von C-Programmen ist es günstig, den symbolischen Debugger SID86 zu nutzen, da die beschriebenen Befehle mit Namen und Symbolen spezifiziert werden können. In der Symboltabelle sind globale Variablen und alle vom Nutzer definierten Funktionsnamen enthalten, die nicht „static", das heißt nur im Übersetzungsmodul bekannt sind. Statische Funktionsnamen werden durch den Namen der Quelldatei spezifiziert, versehen mit der Nummer ihres Auftretens in dieser Datei.

Beispiel:
g,.c.c2

Mit diesem Kommando wird das zweite Unterprogramm der Quelldatei *c.c* erreicht. Außerdem ist es günstig, wenn vom zu testenden Programm mit Hilfe des Reassemblers (Option –r) ein Listing angefertigt wurde, das die C-Anweisungen mit den dazugehörigen Assemblerbefehlen enthält. Das erleichtert das Verfolgen des Programmablaufes.
Nach dem Aufruf des symbolischen Debuggers und dem Laden der Symboldatei kann das Programm im Einzelschrittbetrieb abgearbeitet werden. Adressen und Werte, die in der Symboltabelle enthalten sind, können dabei durch *.name* angegeben werden.
Der Instructionpointer IP steht auf der Startadresse. Die hier stehenden Instruktionen haben aber keine Ähnlichkeit mit den Befehlen des Disassemblerlistings. Es handelt sich hier um die in Abschnitt 10.1.4. erwähnten Funktionen, die vor Starten des Programms *main()* ausgeführt werden müssen.

Beispiel:
g,.MAIN

Mit Hilfe dieses Kommandos gelangt man nun an die Stelle, wo die eigentliche Abarbeitung des C-Programms beginnt. Der Datenbereich, der mit der Adresse *name* beginnt, kann durch das Kommando
d.name
im aktuellen Datensegment angezeigt werden. Es ist auch möglich, Variablennamen mit Registern zu indizieren.

Beispiel:
des:.x+bp

Es wird der Bereich im Extrasegment angezeigt, der mit der Adresse x, vergrößert um den Inhalt von bp, beginnt.
So ist ein recht komfortables Testen des Programms möglich. Allerdings ist dabei die Kenntnis der Assemblersprache des Prozessors Voraussetzung. Wenn diese Bedingung nicht erfüllt ist, muß das Testen des Programms durch zweckmäßiges Einfügen von Ausgabeanweisungen mit den jeweils interessierenden Daten erfolgen. Das ist jedoch recht aufwendig, weil zwischen zwei Tests der vollständige Entwicklungsweg (Editieren, Compilieren, Linken) notwendig ist.

10.2. Systemabhängigkeit des C-Programms

Die Programmiersprache **C** ist auf vielen verschiedenen Systemen implementiert worden. Daraus ergeben sich unterschiedliche Varianten der Umsetzung der Hochsprache in die entsprechende Maschinensprache. Für verschiedene spezielle Anwendungsfälle ist es notwendig zu wissen, wie das übersetzte C-Programm aussieht.

10.2.1. Datenelemente

Die Tafel 10.1 gibt eine Übersicht über die Implementierung der verschiedenen Datentypen. Es ist zu beachten, daß die Bezeichnungen *"short"*, *"long"* und *"unsigned"* nur auf den Datentyp *int* angewendet werden können. Das bedeutet beispielsweise, daß der Datentyp *unsigned long* nicht implementiert ist. Außerdem ist, abweichend vom Standard, *char als unsigned* implementiert. Den Datentyp *enum* gibt es nicht. Der Datentyp *void* wird aus Kompatibilitätsgründen in der Header-Datei *PORTAB.H* durch einen Kommentar definiert. *float* definiert eine 32 Bit lange, vorzeichenbehaftete Gleitpunktzahl mit einem Exponenten der Länge 8 Bit und einer Mantisse von 24 Bit Länge in Hidden-Bit-Darstellung. Der Offset des Exponenten beträgt 127. Dieser Datentyp realisiert eine Genauigkeit von etwa 6 bis 7 Dezimalstellen. Der Datentyp *double* definiert eine 64 Bit lange vorzeichenbehaftete Gleitpunktzahl mit 11 Bit Exponent und 53 Bit Mantisse in Hidden-Bit-Darstellung. Das bedeutet eine Genauigkeit von 15 bis 16 Dezimalstellen. Diese interne Darstellung ist äquivalent der vom 8087-Arithmetikprozessor verwendeten Speicherdarstellung von Gleitpunktzahlen.

Tafel 10.1. Datentypen des DRC-C-Compilers

Typ	Bits	Bereich
char	8	0 bis · 255
int	16	−32768 bis +32767
short	16	−32768 bis +32767
unsigned	16	0 bis · 65535
long	32	$-2 \cdot 10^9$ bis $2 \cdot 10^9$
float	32	$\pm 10^{-37}$ bis $\pm 10^{37}$
double	64	$\pm 10^{-307}$ bis $\pm 10^{307}$

Wenn *Float*-Variablen in C-Programmen verwendet und an Unterprogramme übergeben werden, erfolgt vor der entsprechenden Berechnung eine Umwandlung in den Datentyp *double*, das heißt, die Berechnung erfolgt immer mit höherer Genauigkeit, und erst bei der Abspeicherung des Ergebnisses wird auf die ungenauere Darstellung gerundet.
Pointer auf verschiedene Datentypen sind je nach gewähltem Speichermodell 2 oder 4 Byte lang.
Es können beliebig viele Daten vom Typ *integer* als *register* vereinbart werden. Davon wird die erste im Register SI und die zweite in DI gespeichert. Alle weiteren Registervereinbarungen werden wie *Auto*-Variablen behandelt und entsprechend im Stack abgelegt.

10.2.2. Speicheradressierungsmodelle

Die segmentierte Architektur des 8086 bringt spezielle Probleme für die Implementierung einer Hochsprache mit sich. Um dem Programmierer die Möglichkeit zu geben, eine Kombination von Adressierbarkeit und Effizienz zu wählen, unterstützt der DRC-Compiler zwei Speicheradressierungsarten (auch Speichermodelle genannt).

- Das Small-Modell

Ein kleines Programm, das nur einen geringen Speicherbedarf hat, kann innerhalb eines einzigen Segments plaziert werden. Wenn auch die Menge der zu bearbeitenden Daten die Größe von 64 KByte nicht übersteigt (einschließlich Stack), sind zum Aufruf von Funktionen bzw. für die Adressierung von Speicherplätzen nur die Offsets der vollständigen Adressen erforderlich. Für einen Pointer wird ein 16-Bit-Wort reserviert. Die –b-Option ist in diesem Falle *nicht* gesetzt.
Der Compiler schafft nur zwei Segmente mit je bis zu 64 KByte Länge: **CODE** und *DATA*. Vor Beginn der Abarbeitung des Hauptprogramms wird das Codesegmentregister CS mit der Segmentadresse geladen, und den Segmentregistern DS, ES und SS wird **DATA** zugewiesen. Diese Werte bleiben dann während der gesamten Laufzeit des Programms unverändert. Der Speicher wird für das Small-Modell gemäß Bild 10.1 aufgeteilt.

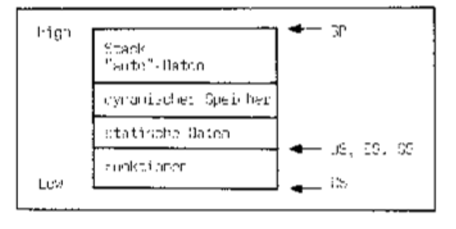


Bild 10.1 Speicheraufteilung im Small-Modell

- Das Large-Modell

Ist das Programm umfangreicher bzw. sind größere Datenmengen zu verarbeiten, ist es notwendig, die absolute Speicheradressierung zu verwenden, das heißt, für jeden Speicherzugriff und Funktionsaufruf muß die vollständige Adresse, bestehend aus Segment und Offset, angegeben werden. Zu diesem Zwecke gibt es das Large-Modell, das durch Setzen der –b-Option ausgewählt wird. Statische und dynamische Daten werden in verschiedenen Segmenten abgelegt. Ein Pointer ist in dieser Adressierungsart 4 Byte lang. Die Aufteilung des Speichers erfolgt nach Bild 10.2.

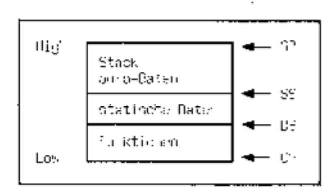


Bild 10.2 Speicheraufteilung im Large-Modell

10.2.3. Funktionsrealisierung

In diesem Abschnitt wird dargestellt, wie Funktionsaufrufe vom C-Compiler realisiert werden.
Wenn ein C-Programm eine Funktion aufruft, werden zunächst die Werte der Argumente auf dem Stack abgelegt, und dann wird der Funktionsruf (CALL) ausgeführt. Das Abkellern der Argumente erfolgt in umgekehrter Reihenfolge (von rechts nach links), so daß sie der aufgerufenen Funktion in der natürlichen Reihenfolge zur Verfügung stehen (von links nach rechts mit steigenden Adressen).

Beispiel:
Der Aufruf der Funktion *func(x,y,z)* mit den drei Integerargumenten x, y und z erzeugt eine Stackaufteilung nach Bild 10.3.
Von der gerufenen Funktion werden dann die folgenden Aktionen ausgeführt:
① Das BP-Register wird auf dem Stack abgelegt und so der Wert des aufrufenden Programms gesichert.
② Das Basisregister BP wird mit dem Inhalt des Stackpointers SP geladen, um eine Adressierung der automatischen und temporären Variablen auf dem Stack mit Hilfe von BP zu ermöglichen.
③ Verwendet die aufgerufene Funktion *Register*-Variablen, werden die Register SI und DI in den Stack gespeichert.
④ Der Stackpointer SP wird um *die* Bytezahl verringert, die von den automatischen Variablen der aufgerufenen Funktion benötigt werden. Dieser Stackbereich umfaßt alle *Auto*-Daten, die in der Funktion deklariert wurden und kann auch einen zusätzlichen Bereich für temporäre Speicherplätze, die oftmals bei der Lösung von Gleichungen benötigt werden, enthalten. Wenn keinerlei automatische oder temporäre Variablen von der aufgerufenen Funktion verwendet werden, wird dieser Schritt übersprungen.
Der Stack hat dann die Anordnung nach Bild 10.4.
Die Adressierung der Argumente und der automatischen Daten erfolgt immer relativ zu BP.
Bevor eine Funktion zur aufrufenden Funktion zurückkehrt, wird der Rückgabewert (falls vorhanden) in bestimmte vorgegebene Register geladen. Welche Register dabei benutzt werden, wird durch das Format des Rückgabewertes entsprechend der folgenden Tafel bestimmt.

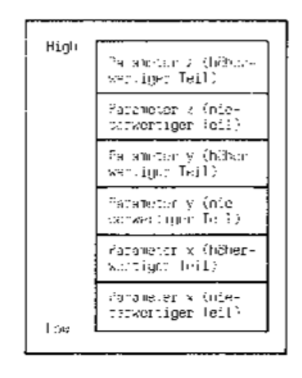


Bild 10.3 Anordnung der Funktionsparameter im Prozessorstack

Länge des Rückgabewertes	Register	Beispiel
8 Bit	AL	char
16 Bit	AX	int, Pointer im Small-Modell
32 Bit	(BX, AX)	long, float, Pointer im Large-Modell
64 Bit	(DX, CX, BX, AX)	double

Wenn mehrere Register verwendet werden, enthält *AX* den *niederwertigsten* Teil des Ergebnisses. Das heißt, wenn im Large-Modell ein Pointer zurückgegeben wird, enthält BX die Segmentadresse und AX die Offsetadresse.
Aus der Organisation der Übergabe des Funktionswertes geht hervor, daß *keine Strukturen* an die aufrufende Funktion zurückgegeben werden können. Als *Argument von Funktionen* sind Strukturen dagegen zugelassen. Sie werden vollständig in den Stack übertragen.
Wenn die Funktion beendet wird, erfolgt zunächst die Rücksetzung des SP um den Bereich der automatischen Variablen und nach dem Rückretten eventuell gekellerter SI- und DI-Register und des Basepointers BP erfolgt

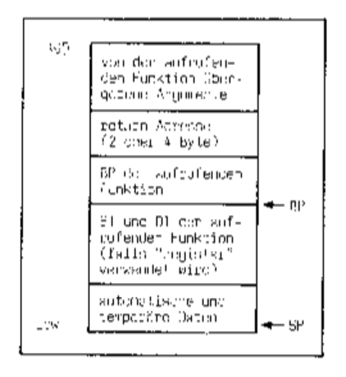


Bild 10.4 Inhalt des Stack bei Ausführung der Funktion

je nach Speichermodell ein NEAR- oder FAR-Return. Weil eine C-Funktion in besonderen Fällen auch eine unterschiedliche Zahl von Argumenten haben kann (z. B. printf()), kann die aufgerufene Funktion den von den Parametern belegten Speicherplatz nicht freigeben (d. h. der Befehl *ret n* wird nicht verwendet), sondern dieses Rücksetzen des SP wird von der aufrufenden Funktion durchgeführt.

10.2.4. Einbindung maschinen- bzw. betriebssystemabhängiger Programmanteile in C-Programme

In einigen Fällen der systemnahen Programmierung erweisen sich die in den Laufzeitbibliotheken enthaltenen Standardfunktionen als nicht ausreichend. Das ist speziell dann der Fall, wenn maschinenspezifische Programme abgearbeitet werden sollen, wie es beispielsweise bei Portein-/Portausgaben und beim Einbinden spezieller 8087-Programme erforderlich ist.

• Verbindung von Assembler- und C-Programmen
Es ist möglich, Assemblermodule für die Einbindung in C-Programme zu schreiben, da der Assembler RASM86 ein gleich aufgebautes Objektfile liefert (siehe Abschnitt 6) und der Linker LINK86 mehrere solcher Objektdateien verbinden kann. Allerdings sind beim Erzeugen solcher Assemblerdateien verschiedene Besonderheiten zu beachten:
① Der Name des von einer C-Funktion aufzurufenden Assemblerprogramms muß in der A86-Datei als PUBLIC vereinbart werden, damit er dem Linker bekannt ist.
② Da der Assembler alle Buchstaben als Großbuchstaben behandelt, muß der Name der Funktion auch im C-Programm mit großen Buchstaben geschrieben werden.
③ Die Auswertung der vom C-Programm übernommenen Parameter muß den unter 10.2.3. beschriebenen Konventionen für die verschiedenen Speichermodelle entsprechen. Das bedeutet insbesondere, daß Funktionen, die mit CALLF aufgerufen wurden (Large-Modell), mit RETF abgeschlossen werden. Auch ist der unterschiedliche Offset der übergebenen Parameter relativ zu BP – bedingt durch die unterschiedliche Länge der Return-Adresse – bei den verschiedenen Speichermodellen zu beachten.
④ Wenn ein Funktionswert zurückgegeben werden soll, muß er in die entsprechenden Register (siehe Punkt 10.2.3.) eingetragen werden.

```
          CSEG
INP:      PUSH BP
          MOV BP,SP
          MOV DX,X[BP]
          IN AL,DX
          XOR AH,AH
          POP BP
;
IF MODEL
          RETF
ELSE
          RET
ENDIF
```

• Einbinden von CP/M-BDOS-CALLs
Viele der implementierten Standardfunktionen nutzen die BDOS-CALLs des Betriebssystems (z. B. Tastaturein-/Tastaturausgabe). Es ist aber auch möglich, in Anwenderprogrammen BDOS-CALLs zu nutzen, falls die Standardfunktionen nicht ausreichen. Das geschieht mit Hilfe der Funktion *_BDOS(nr,value)*.
Der Parameter *nr* wird in das Register CX geladen und spezifiziert die Nummer des BDOS-Calls. Der Wert *value* wird in das Register DX übernommen.
Die Funktion *__BDOS()* hat folgende Wirkung: Die externe Variable *_cpmrvcx* enthält den vom BDOS-CALL in CX zurückgegebenen Wert, und in *_cpmrv* wird der Wert des Registers AX geladen. Dieser Wert ist natürlich auch als Funktionswert der Funktion verfügbar. Das folgende Beispiel zeigt die Einbindung der Abfrage des Tastaturstatus in ein C-Programm mit Hilfe von BDOS-CALLs.

Beispiel:

```
#include <std.h>
main()
{
  char c;
  ...
  if(__BDOS(11)){
  /* Abfrage des Tastaturstatus */
  /* Taste gedrückt */
     ...
  } else {
  /* Taste nicht gedrückt */
     ...
  }
  ...
}
```


Literatur

/1/ Dokumentation für Echtzeitbetriebssystem BOS 1810. VEB Kombinat Robotron
/2/ Claßen, L.; Oefler, U.: UNIX und C. VEB Verlag Technik Berlin: 1987
/3/ Horn, T.: Programmierung in C. Mikroprozessortechnik. Berlin 1(1987) 1–6
/4/ Kernighan, B. W.; Ritchie, D. M.: Programmieren in C. München, Wien: Carl-Hanser-Verlag, 1983
/5/ Dokumentation SCP 1700, VEB Robotron Elektronik Dresden

Schluß

TERMINE

VIDEOTON-Ausstellung

WER? VIDEOTON
WANN? 24. bis 28. April 1989
WO? Botschaft der Ungarischen Volksrepublik in Berlin
WAS?
• Rechentechnik
• Robotertechnik
• Fernsehgeräte
• Nachrichtentechnik
WIE? Ausführliche Informationen über das Programm erhalten Sie in den VIDEOTON-Vertretungen der DDR: Handelsvertretung der Botschaft der UVR, Tel. 2202561, Kundendienstbüro Berlin, Tel. 4724185/86, Kundendienstbüro Erfurt, Tel. 71 1472.

Görs

Hinweis

Im Beitrag „Indizierte Variablen und Felder unter REDABAS" von Dr. Thomas Streubel in MP 3/89, S. 89, muß es in der zweiten Spalte in den Zeilen 4 und 5 richtig heißen:
store substr(str((&ind+1001),4,0),2,3) to ind (16 Bit)
Autor und Redaktion bitten, den Fehler zu entschuldigen.